SUMMER PROCEEDINGS 2018

The Center for Computing Research at Sandia National Laboratories

Editors:
Attila Cangi and Michael L. Parks
Sandia National Laboratories

December 6, 2018

SAND#: SAND2019-5093 R

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

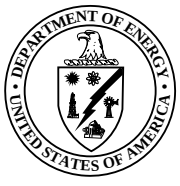
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@osti.gov
Online ordering:	http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.gov
Online ordering:	http://classic.ntis.gov/help/order-methods/



Preface

The Center for Computing Research (CCR) at Sandia National Laboratories organizes an active and productive summer program each year, in coordination with the Computer Science Research Institute (CSRI) and Cyber Engineering Research Institute (CERI). CERI focuses on open, exploratory research in cyber security in partnership with academia, industry, and government, and provides collaborators an accessible portal to Sandia's cybersecurity experts and facilities. Moreover, CERI provides an environment for visionary, threat-informed research on national cyber challenges. CSRI brings university faculty and students to Sandia National Laboratories for focused collaborative research on DOE computer and computational science problems. CSRI provides a mechanism by which university researchers learn about problems in computer and computational science at DOE Laboratories. Participants conduct leading–edge research, interact with scientists and engineers at the laboratories, and help transfer the results of their research to programs at the labs.

Within the CCR Summer Program students from around the country conduct internships at Sandia. Each student is paired with a Sandia staff member who serves as technical advisor and mentor. The goals of the summer program are to expose the students to research in the mathematical and computer sciences at Sandia and to conduct a meaningful and impactful summer research project with their Sandia mentor. Every effort is made to align summer projects with the student's research objectives and all work is coordinated with the ongoing research activities of the Sandia mentor in alignment with Sandia technical thrusts.

The CCR Summer Program consists of two key components, the CCR Summer Seminar Series and the CCR Summer Proceedings.

All summer students and their mentors are encouraged to contribute a technical article to the CCR Summer Proceedings. In many cases, the CCR Proceedings are the first opportunity that students have to write a research article. Not only do these proceedings serve to document the research conducted during the summer but, as part of the research training goals of Sandia, it is the intent that these articles serve as precursors to or first drafts of articles that could be submitted to peer-reviewed journals. As such, each article has been reviewed by a Sandia staff member knowledgeable in that technical area with feedback provided to the authors. The contribution for the 2018 CCR Proceedings have been organized into the following broad technical focus areas—*Software and High Performance Computing, Computational Mathematics, Applications* We would like to thank all participants who have contributed to the outstanding technical accomplishments of CSRI and CERI in 2018. The success of the program hinged on the hard work of enthusiastic students and their dedicated Sandia technical staff mentors. We would also like to thank those who reviewed articles for this proceedings; their feedback is an important part of the research training process and has significantly improved the quality of the reports.

An important educational component of the summer program is the CCR Summer Seminar Series. We would like to thank the staff who gave presentations at the 2018 Series: Tim Wildey (1441) on "Combining Measure Theory and Bayes' Rule to Solve a Stochastic Inverse Problem", Allen Robinson (1443) on "Z, Z-Next and Multiphysics modeling: Opportunities in plasma physics modeling", Reed Milewicz (1446) on "Software Engineering for Science: Challenges, Opportunities, and Research Perspectives", Bill Rider (1446) on "The History of Computational Fluid Dynamics, Part I: CFD Before CFD", Marta D'Elia (1441) on "Nonlocal Models in Computational Science and Engineering: Challenges and Applications", Ken Moreland (1461) on "Building Better Plots", Steve Plimpton (1444) on "Particles, HPC, and the Ukulele Syndrome", Luke Shulenburger (1641) on "Towards

Ab initio Materials Predictions with Known Accuracy", and Andrew Baczewski (1425) on "Light-matter interaction in the extremes of time, intensity, and energy".

Finally, the CCR summer program would not be possible without the administrative support of Christine Trujillo, Steven Garcia, Celia Montoya, Ashley Avallone, Sandy Portlock, John Perseo, Emily Lujan, and especially Lorena Martinez.

Attila Cangi
Michael L. Parks

December 6, 2018

Table of Contents

Software and High Performance Computing

The articles in this section discuss the implementation of high performance computing (HPC) and productivity software. In many cases, performance improvements and portability are demonstrated for many-core architectures, such as conventional multicore CPUs, the Intel Many Integrated Core coprocessor (MIC), and graphical processing units (GPU).

Sahasrabudhe, Phipps, and Rajamanickam address code portability across wide range of architectures by extending previous work on leveraging single instruction multiple data (SIMD) type primitive with overloaded operators. They provide a GPU support for SIMD type primitives and compare performance characteristics on different types of kernels from real life scientific applications such as general dense matrix-matrix multiplication and char oxidation kernel of Uintah.

Hutter and Rajamanickam develop the TCKK software package, a library for performance-portable batched tensor contractions in highly parallel MIC architectures and GPUs. Its applications include finite element codes and convolutional neural network kernels. They present its interface and assess performance results on the Intel Knights Landing architecture.

Yasar, Rajamanickam, Wolf, Berry, and Çatalyürek improve upon the general sparse matrix-matrix multiplication (SpGEMM) algorithm in the Kokkos kernels library which is triangle counting algorithm for graph analysis. They demonstrate significant speed-ups in runtime compared to the previous version and to traditional graph based formulations.

Strack, Luchini, Elliott, and Siefert discuss methods to reduce the amount of runtime memory in the ALEGRA shock and multiphysics computational simulation code for problems beyond 2^{32} topological entities in a finite element mesh. They realize this with new compiler and linker options, as well as by transforming the code base from 32-bit to 64-bit using a new C++ class to reference its finite elements.

Yenpure and Moreland present a fast and efficient approach to merging three-dimensional points in space based on data parallel techniques in shared memory environments. They demonstrate the scalability and efficiency of their approach by comparing against widely used scientific visualization libraries on parallel hardware such as many-core CPUs and NVIDIA GPUs.

Morris and Curry address the problem of disk failure prediction using machine learning. They train their machine learning model on raw S.M.A.R.T. attributes from a dataset over a vast amount of disks and models from various manufacturers. Performing with a very low false positive rate, their model is a practical tool for anticipating disk failure in real situations.

Chen, Lofstead, and Mitchell develop the *stitch* library, an efficient I/O API and database format that leverages novel data storage and I/O methods to accelerate multiscale modeling of manufacturing materials. They achieve comparable scaling performance and increased data storage flexibility to other I/O methods. They demonstrate this by applying their *stitch* library to kinetic Monte Carlo simulations of additive manufacturing using *spparks*.

A. Cangi
M. L. Parks

December 6, 2018

PERFORMANCE PORTABLE SIMD SCALAR TYPE FOR EFFECTIVE VECTORIZATION ACROSS HETEROGENEOUS ARCHITECTURE

DAMODAR SAHASRABUDHE*, ERIC PHIPPS†, AND SIVASANKARAN RAJAMANICKAM‡

Abstract. To achieve exascale computing, new and upcoming architectures of CPUs are getting equipped with more and more capable vector units along with longer vector lengths. Hence effectively using these vector units becomes a key. At the same time, the architectures are evolving rapidly. Heterogeneous and/or many core architectures are being deployed in largest supercomputers today. Thus the code portability across wide range of architectures has become all the necessity of the time.

This study extends the previous work on leveraging "SIMD" type primitive with overloaded operators which is allows users to effectively use intrinsics without compromising portability. This work provides a GPU support for "SIMD" type primitives which was not present so far and compares performance characteristics on different types of kernels from real life scientific applications such as GEMM and char oxidation kernel of Uintah. The ultimate goal is to add this capability into kokkos so that it can be easily used by wider scientific community.

1. Introduction. With efforts for exascale computing, the usage of vector units is becoming more and more important. Each generation of processors is adding new capabilities in Vector Processing Units (VPUs) coupled with increase in vector length and memory bandwidth that can support massive data movement. Intel's KNL supports vector length of 512 bits, while upcoming ARM processors are supposed to have vector length of 2048 bits.

Thus efficiently utilizing these vector units has become a key to extract the performance from new architectures. However using SIMD on data parallel tasks is not as easy as it would seem. Traditionally compilers fail to auto vectorize the code if it has complex control flow such as nested if conditions or nested loops or break statements. The alternative to efficiently utilize SIMD capabilities is to use architecture dependent vector intrinsics. Although intrinsics can deliver near optimal performance, it makes code non-portable across different architectures. Few studies have been carried out to remedy the problem using a "SIMD" as a primitive data type and providing operator overloads to carry on arithmetic operations on "SIMD" data [1, 3, 5, 6]. The basic idea in all the works is same - Declare a primitive (say a class or a structure) containing an array or a intrinsic SIMD datatype. Thus each instance of the primitive contains elements equal to vector length. Each of them also contains overloaded operators which operate on simd data in primitive. These overloaded operators process primitive elements using platform supported intrinsics. For each new platform new set of intrinsic back end and primitive data type created. Thus when users use a primitive or map an array to a primitive, each instance of it can execute operation on elements of primitive in SIMD fashion.

However none of these implementations have support for GPUs at the moment. This restricts use of these "SIMD" premitives in portable implementations such as those using Kokkos. Kokkos is a performance portable library designed to provide consistent front end interfaces to users while Kokkos backend is adapted as per the architectures. Further Kokkos provides different scheduling policies to extract maximum performance out of underlying architecture achieving performance portability. Kokkos kernels has it's own version of this "SIMD" data type, but without support for GPUs [2].

Pertaining to different nature of kokkos-kernel use case and stokhos [5] use case, both have different implementation of "SIMD" type. This works aims to provide GPU support for SIMD primitive and also create a generic SIMD primitive with flexible vector length

*University of Utah, damodars@sci.utah.edu
†Sandia National Laboratories, etphipp@sandia.gov
‡Sandia National Laboratories, srajama@sandia.gov

independent of physical vector length provided by underlying hardware. Further it desires to support both uses cases of kokkos kernels and stokhos. The ultimate goal of the study is to include this "portable SIMD primitive" into kokkos and release it for all the kokkos users allowing larger scientific community to effectively take advantage of vector units.

2. SIMD primitive for CPU. Following pseudo code shows simplified version of existing SIMD primitive in kokkos-kernel for KNL:

```
template <typename T> struct Vector;

//specialization for double
inline template<> struct Vector<double> {
 __m512d _data; //KNL 512 bits datatype for double

 //different constructors to initialize _data
 .
 .

 //overloaded operators
  Vector& operator= (const Vector& x) {
   _data = x._data;
   return *this;
 }
};

 //overloaded operators
 inline Vector operator+ (const Vector& a, const Vector& b) {
  return Vector(_mm512_add_pd(a._data, b._data));
 }

 //overloaded math library functions
inline Vector sqrt(const Vector& x) {
 return Vector(_mm512_sqrt_pd(x._data));
}
```

Listing 1: Simplified SIMD primitive

"Struct Vector" is a structure template used for SIMD primitive data type. Users can pass the required datatype as a template argument. "template¡¿ struct Vector¡double¿" shows the specialization for data type double. Each datatype is implemented as a specialization. For double, `_m512d` is used as underlying datatype, which is a simd data type provided by intel with length of 512 bits. The sturcture provides different constructors (not shown in the code) which can be used to initialize class with variety of different ways such as from a scalar variable (double), from another simd type (i.e. Vector¡double¿) or directly from `_m512d`.

Operators = and + a are overloadeded to accept another Vector as an input and return appropriate Vector result depending upon operation. It should be noted that both the operations are performed using vector intrinsics. As the code is inlined by compilers, it appears (and performs) as if written using intrinsics. Similarly math functions are also overloaded which in turn call in built functions operating on intrinsics.

Similar implementations are present in stokhos and stk packages.

3. Use Cases. The portable implementation of SIMD type tries to achieve three important uses case:

- Kokkos kernels: The general dense matrix-matrix multiplication (Gemm) and triangular solve (TRSM) kernels implementations within kokkos kernels operate on multiple matrices. It rearranges data in "compact batch layout" where matrix dimension is the smallest dimension i.e. elements are placed as (row, column, matrix) rather than traditional (matrix, row, column) [2]. Thus elements of same cell given by same (row, column) of all matrices are placed adjacently in the memory then comes (row, col+1) and so on. Hence matrix dimension is used as simd dimension. In case of KNL operating on double precision floating point data, each simd operation at a time processes 8 elements in SIMD using SIMD primitives, processing 8 matrices simultaneously. Thus users iterate over matrix loop for "number of matrices / 8" iterations only.
- Stokhos: Stokos implementation of SIMD primitive is similar except it takes "Vector Length" as an additional template parameter. Users pass number of matrices as Vector length assuming each SIMD element contains all the elements across entire matrix dimension. Thus any operation process all the elements in the matrix dimension by internally iterating over "number of matrices / 8" iterations. This simplifies the user code as it makes entire matrix dimension to disappear from users' context. Users simply need to perform operations as if operations are done on scalar type without bothering about matrices.
- Char oxidation in Uintah [4]: Small gemm and trsm kernels in kokkos-kernels are straight forward and quite easy to tune in such a way that compiler can auto vectorize. To evaluate real potential of SIMD type, it is necessary to test it on real life large and complex scientific application kernel which has enough control flow which certainly prevents compiler auto-vectorization. Uintah provides such an opportunity. Uintah is a massively parallel multi task run time system used to solve PDEs in order to simulate complex physics problems such as combustion. Uintah is being used for simulation of a coal boiler under DoE's PSAAP II project and has been successfully scaled across 256k, 512k cores on MIRA and Titan respectively. The Arches component of Uintah used to simulate combustion has a kernel which simulates char oxidation. The kernel iterates over each cell of a 3 dimensional patch. The kernel has around 300 lines of codes. It has computations with if - else conditions, multiple for loops over "number of reactions" (which account for different reactions taking place during the oxidation process) followed by a Newton-Raphson solve to find out rhl. (need to fin out what is rhl) The for loop over Newton-Raphson solver iterations contains few more "reactions" loops and also if and break statements to check the residual and exit the loop. Finally the kernel computes six different quantities for every cell - char mass rate, char rate, gas char rate, particle temperature, particle size and surface rate. This computationally intensive kernel presents a highly data parallel task. However complex control flow in it prevents the auto vectorization by the compiler and forms an ideal candidate to try out effectiveness of SIMD primitives.

4. Portable SIMD primitive. As mentioned in section 1, existing SIMD primitives do not do not support execution on GPU and thus can not be used in portable codes such as kokkos kernels.

The GPU (more specifically CUDA) support for SIMD primitives is added as part of this work. CUDA inherently executes the code in SIMD fashion. Hence adding CUDA support for SIMD primitives will not give any added performance boost on GPU, however it will make primitives truly portable by allowing heterogeneous execution at no additional cost.

Challenges: The main challenge in making SIMD primitive portable is a type of

temporary SIMD variables declared inside kernel. Consider following pseudo code for matrix multiplication using kokkos `parallel_for` assuming row major order:

```
//assuming vector length 8 for KNL
#define VEC_LEN 8

//Vector<double> is a SIMD primitive for double
Vector<double> A[N][N][M/VEC_LEN], B[N][N][M/VEC_LEN], C[N][N][M/VEC_LEN];

//populate A and B.

Kokkos::parallel_for(N*N, [&](int id) { //parall_for across rows and columns
  int i = id/N, j=id%N; //row and column ids.
  for(int m=0; m<M/VEC_LEN; m++) //iterate over all matrices
  {
    Vector<double> temp;
    for(int k=0; k<N; k++)
     temp += A[i][k][m] * B[k][j][m];
   C[i][j][m] = temp;
  }
});
```

Listing 2: Return type challenge

The main reason for using kokkos `parallel_for` is to make it portable and of course TO extract parallelism. A, B and C are arrays of M matrices each of size N x N. These are declared using SIMD primitive - Vector¡double¿ and are laid out with matrix (M) dimension as the smallest dimension so that vectorization can take place across matrices. m th dimension is divided by `VEC_LEN` to get M/`VEC_LEN` chunks, each containing `VEC_LEN` elements. The loop will do the standard multiplication of 'i'th row of A and 'j'th column of B for 'm'th chunk of matrices and accumulate it's result in temp. Note that each element returned by A, B and C is SIMD element and contains 8 doubles. Same is true for temp.

Now when this code is ported to cpu, by kokkos, 'temp' and matrix element both have same length of `VEC_LEN`. The code works smoothly because the declaration of 'temp' is outside simd context and hence each element of temp gets mapped to the vector lane of VPU. However when the same code is ported to GPU, entire code is executed in the SIMD context due to execution model of CUDA. Thus each simd thread declares 'temp' of size `VEC_LEN`. Although each thread operates on it's own simd lane of temp, thereby making code work technically, rest of the elements of temp always remain unused and waste huge amount of memory.

The second challenge is the return type of operation A[i][k][m] * B[k][j][m]. CPU intrinsics return the intrinsic packed element. However each GPU thread acts as a vector lane and returns a scalar element i.e. double. The responsibility to collect data from each thread and map to correct element in SIMD primitive lies on the developer. Hence CUDA implementation of SIMD primitive needs combinations of operands - (Vector, Vector), (Vector, Scalar) and (Scalar, Vector).

Implementation: Several options were tried to address the challenges and the fastest was picked up for testing GEMM and Uintah kernels. Few key approaches discussed below along with performance results for dummy operations (such as A(index) /= (A(index) + ((B(index) + A(index)) + B(index))) + (B(index) + B(index))) on one dimensional arrays repeated for several thousand times show the evolution of the final implementation.

- **Dummy SIMD Type**: In this approach, CUDA SIMD type was written with only

Approach	Execution time (s)
Dummy SIMD Type	0.28
Different Kernel type	0.28
Variable Vector Length, EVL=1	0.28
Variable Vector Length, EVL=2	0.4
Using Shared Memory Pool, EVL=2	7

*EVL: Elements per Vector Lane

Table 4.1: Performance results for different approaches on K80

one element rather than using an array of `VEC_LEN` elements. This solved all the challenges. Because each SIMD element contains only one data element, it exactly maps to value returned by a single CUDA thread. Further declaring it inside SIMD context also works well as it does not waste extra memory space as explained earlier. Further this approached gave the best performance as each element returned a single double, it could well be stored in registers rather than in memory. However problem arises when SIMD primitive is included as element of another class / structure. It occupies space of only one scalar element, where semantics would expect it to occupy space of `VEC_LEN` thus disturbing all the offset calculations.

- **Different Kernel type**: To address the problem posed by Dummy SIMD Type, a new type "SIMD Kernel Type" was introduced. SIMD primitive is set to `VEC_LEN` always and is used to declare variables outside kernel i.e. outside simd context. New SIMD Kernel Type to is utilized to declare variables inside kernel, if needed. For CPU, SIMD Kernel type is same as SIMD primitive. For CUDA, SIMD Kernel Type contains only one scalar element. The approach needs additional operators overloaded as mentioned in the second challenge. Further each thread access it's own elements using threadIdx.x as an index into the data of SIMD primitive. SIMD Kernel Type does not need index because it contains only one scalar element and each thread has it's own copy of it. This approach performs equally good as Dummy SIMD Type as shown in the table 4.1.
- **Variable Vector Length**: Stokhos use case mentioned earlier needs the vector length to be platform independent. To support it, new template parameter "Vector length" is added to "Different Kernel type" approach. It is required that the Vector Length passed should be multiple of the physical vector length of underlying architecture. This basically assigns multiple elements per vector lane or cuda thread. Overloaded operators loop over the SIMD primitive data for "Vector Length / physical vector length" number of iterations. However the disadvantage is that CUDA has to allocate memory space to return / pass multiple elements per vector lane. This memory would naturally be allocated from the global memory rather than registers as in case of previous two approaches. As a result this approach performs slower unless Vector Length is set same as physical vector length where this approach become same as "Different Kernel Type"
- **Using Shared Memory Pool**: To rectify slowdown caused by multiple elements per vector lane in "Variable Vector Length" approach, the return type was explicitly allocated from shared memory pool with goal to avoid long latency global memory access. To avoid bank conflicts, each element within assigned to one CUDA thread was allocated to space offseted by `VEC_LEN`. This allowed all elements of CUDA thread to be placed in a single memory bank and each thread in a warp can access

it's own element with a single load / store cycle. However this method did not perform as expected due to overload of maintaining pool. This overhead increases exponentially as number of operations increase in spite of reusing allocated shared memory chunk. Table 4.1 shows the impact of the pooling overhead.

The first three entries in table 4.1 represent the same underlying structure - one scalar element per vector lane which is passed / returned using registers. All of them perform equally well. When elements per vector lane are increased to two, performance degrades from 0.28 to 0.4 seconds. The version using shared memory pool performed worst taking 7 seconds. Hence out of all the variants, "Variable Vector Length" was chosen for the implementation which is most flexible and delivers best performance when used with one element per vector lane.

5. Experiments. To test the implementation of performance portable SIMD primitive, two different types of kernels were used.

5.1. DGEMM. The performance portable kernel to carry out dense general matrix multiplication over array of M matrices was written using Kokkos. Kokkos TeamPolicy was used to achieve parallelism at multiple levels. Matrices are split equally across teams and each thread team works on it's own chunk. Next rows and columns are split across threads within a team as shown in pseudo code in section 4. Further SIMD primitive is spanned across matrix dimension. Thus each CPU thread (or a CUDA Warp) carries out each operation on `VEC_LEN` elements of `VEC_LEN` matrices in simd fashion. Few more experiments were conducted using Kokkos RangePolicy instead of team policy. Standard tiling optimizations with tile sizes 3 x 3, 4 x 4 and 5 x 5 were introduced to extract spacial and temporal locality among matrix elements. Experiments were carried out using 4 different matrix size - 3 x 3, 5 x 5, 10 x 10 and 15 x 15. Each time 16,384 number of matrices were evaluated.

The portable code was successfully compiled and run on Nvidia Pascal 100 and Volta with Cuda back end and on intel's KNL using OpenMP back end for kokkos *without any code changes*. These results are then compared against results of GEMM kerenls present in Kokkos-Kernels package. As of now kokkos-kernels has two separate kernels for CPU and GPU. Those respective kernels are used for testing.

Different combinations of number of threads (and teams in case of team policy) were tried and the best timing was picked.

5.2. Uintah Kernel. Transforming Uintah [4] kernel was more challenging because of complexities mentioned earlier in use cases. The main steps of vectorization using SIMD primitive are listed below:

- **Cast Uintah data structures**: The first step was to obtain raw data pointer and `reinterpret_cast` to array SIMD Primitives. The casted pointer is then converted into Kokkos unmanaged view based on SIMD primitives. Further subview is taken to offset as per patch boundaries. This subview is then embedded into KokkosView3 data structure within in Uintah. This Kokkos view is then used everywhere in the kernel instead of existing scalar kokkos views.
- **Adjust iterations and array dimension**: SIMD execution is supposed to take place across cells of a patch in x dimension. Thus number of iteration in x dimension of `paralle_for` and array dimensions in x dimension are both divided by `VEC_LEN`. As of now number of cells in x dimension are assumed to be multiple of `VEC_LEN` and hence explicit scalar iteraiton for remainder loop are not added.
- **Replace local variable**: Each local variable declared *inside* the kernel before operated on a single cell and was a scalar. However with simd execution, each iteration would process `VEC_LEN` number of cells in parallel and hence each local

variable needs to hold `VEC_LEN` scalars. Hence it is replaced with "SIMD Kernel Type". With these 3 changes, most of the computations got aligned with simd execution. However handling control flow was tricky part.

- **for loops over Number of Reactions**: The kernel computes effects of multiple chemical reactions taking place in boiler and several arrays of length "Number of Reactions" are declared inside kernel and are used in multiple for loops scattered all over the kernel. To use SIMD primitive, each array was changed to SIMD primitive type instead of scalar type. The for loop control remains unchanged. Thus each iteration now executes for loop over number of reaction in SIMD style across patch cells due to operators overloaded in SIMD primitive.
- **Solver Iterations**: The kernel solves the oxidation equation (Need to confirm which equation) using Newton-Raphson method. Depending upon residual value break statement comes of out iteration loop and the number of solver iteration vary for each cell. Thus this becomes a major hurdle in SIMD execution - especially for compiler to auto vectorize. To overcome this problem, a scalar loop separately tests whether the equation is solved at the end of each solver iteration. As long as at least one cell is not converged the solver loop continues. However this overwrites solutions of those which are already converged. To fix this error a temporary SIMD variable is used to store results as soon as solution for the cell converges. This temporary variables remains untouched later because the scalar loop ignore the cell once it is converged. As soon as all solutions for all cells converge, solver loop exits and values from temporary solution variable are written back to real solution which is used in the subsequent code. Although this technique makes the solve time of entire group to solve time of slowest cell, it still proves faster than scalar solve of individual cells.
- **If else**: Typical if else computations done using intrinsics use masked intrinsics for conditional evaluation. However it is not an option in this case as operators can not take third parameter for mask. Declaring global variable for mask would not be such an elegant idea. Preferred alternative was to evaluate if condition and cast it into an integer 0 / 1 and multiply the result with this integer value. The disadvantage is it forces calculation of if and else both, but still proved to be faster than scalar code on KNL. Some if else used global values in conditions rather cell based variables. This easier as control flow would remains same for entire simd and converting contents of if - else was enough.

The experiments were carried out on intel's KNL using two different domain sizes - 64 cubed and 128 cubed. Every time, domain was divided among 64 patches leading to two different patch sizes - 16 cubed and 32 cubed. 64 patches were distributed among 4 ranks each rank spawned 4 teams of threads with 4 threads per team thus assigning 64 threads to 64 cores of KNL. Each team is assigned with multiple patches. The z dimension of each patch is parallelized among threads within a team and each thread executes x dimension in simd fashion.

6. Results.

6.1. GEMM Kernel. Figure 6.1 compares GFLOPS of existing GEMM kernel in Kokkos-kernels package with new portable kernel written using kokkos and new SIMD type primitive and run on Nvidia Pascal 100 and intel's KNL respectively.

As expected, KNL results of SIMD primitive show similar GFLOPS to Kokkos-kernels as Kokkos-kernels version is already vectorized and has all optimizations such as tiling.

Similarly the performance of CUDA should be remain same for CUDA version of kokkos kernels and for kernel using SIMD type primitive, because CUDA inherently executes in SIMD mode and SIMD scalar type does not make provide any extra advantage. The algorithm and

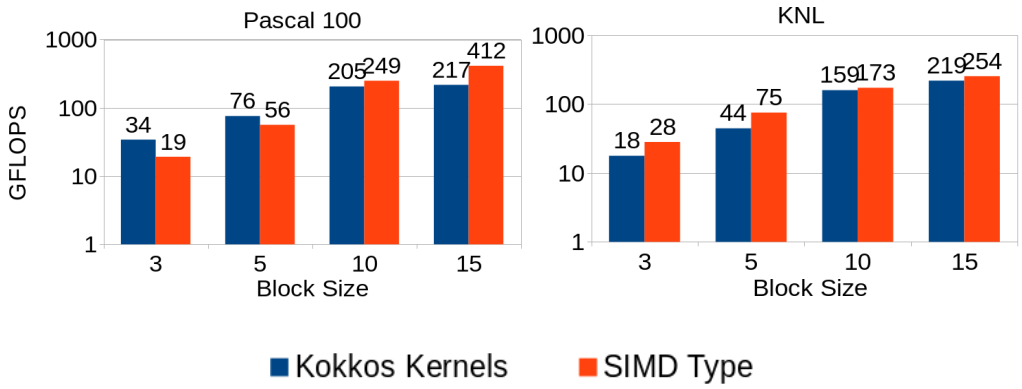


Fig. 6.1: Comparison of GFLOP between Kokkos Kernels and SIMD Type

optimizations used in both are also same. However 1.7x slowdown was observed for matrix size of 3. The performance of SIMD primitive gradually improved with matrix size and for SIMD primitive ran 2x faster than kokkos kernels for matrix size of 15. This particular behaviour demands further investigation. Few possible reasons are:

- Among versions, range policy version is the fastest for matrix sizes 3 and 5. On the other hand SIMD Primitive used team policy. Possibly overhead of team is not justified fo such a small matrix size. Need to test Range policy version of SIMD primitive.
- Kokkos kernels spawn 1024 teams of 64 threads per team with vector length of 16 for matrix size 15. Thus number of threads spawned = 1024 x 64 x 16 = 1 million. This possibly creates an overhead and not enough work per thread especially when tiling is used. On the other hand SIMD Primitive version initiated only 1024 x 3 x 16 = 48k threads.

	16 cubed patch	**32 cubed patch**
Scalar	13	100.4
Vector	2	12.76
Speedup	6.5x	7.8x

Table 6.1: Uintah's Char Oxidation Kernel Execution time (ms) on KNL

6.2. Uintah Char Oxidation Kernel. Uintah results gave more spectacular results on KNL than GEMM kernel. Char oxidation kernel vectorized using SIMD primitive type showed speedups of 6.5x and 7.8x compared with existing scalar version for patch size 16 cubed and 32 cubed respectively as shown in table 6.1. It is worthwhile to note vectorization became possible due to SIMD primitive type where auto vectorization by compiler was not possible earlier due to complex control flow.

7. Conclusion. The SIMD primitive type proved to be easy to use and yet very efficient vectorization technique as was done in previous studies. It is especially handy when the code has complex control flow and compiler refuses to auto vectorize giving near optimal speedups.

The more important aspect of this work is adding CUDA support for SIMD primitives at no extra overhead compared to code written using plain CUDA. It makes the SIMD Primitives performance portable when used along with kokkos.

8. Work In Progress. This is work in progress / future work:

- Need to verify the performance difference - Possible reason of slowdown for matrix sizes 3 and 5 is team policy. Need to try range policy. Also verify theory of thread count for improved performance for matrix sizes 10 and 15.
- Implement DGETRF kernel.
- Try running CUDA version of Uintah kernel.
- Add the code in Kokkos source code and carry out thorough testing making it ready for kokkos team.

9. Acknowledgements. Authors are thankful to Christian Trott and Simon David Hammond from Sandia National Labs and John Holmen and Martin Berzins from the University of Utah for their advice and help.

REFERENCES

[1] P. Karpiński and J. McDonald, *A high-performance portable abstract interface for explicit simd vectorization*, in Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores, ACM, 2017, pp. 21–28.

[2] K. Kim, T. B. Costa, M. Deveci, A. M. Bradley, S. D. Hammond, M. E. Guney, S. Knepper, S. Story, and S. Rajamanickam, *Designing vector-friendly compact blas and lapack kernels*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 55.

[3] M. Kretz and V. Lindenstruth, *Vc: A c++ library for explicit vectorization*, Software: Practice and Experience, 42 (2012), pp. 1409–1430.

[4] Q. Meng, A. Humphrey, and M. Berzins, *The uintah framework: A unified heterogeneous task scheduling and runtime system*, in Digital Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis, 2012, pp. 2441–2448. SC'12 –2nd International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing, WOLFHPC 2012.

[5] E. Phipps, M. D'Elia, H. C. Edwards, M. Hoemmen, J. Hu, and S. Rajamanickam, *Embedded ensemble propagation for improving performance, portability, and scalability of uncertainty quantification on emerging computational architectures*, SIAM Journal on Scientific Computing, 39 (2017), pp. C162–C193.

[6] H. Wang, P. Wu, I. G. Tanase, M. J. Serrano, and J. E. Moreira, *Simple, portable and fast simd intrinsic programming: generic simd library*, in Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing, ACM, 2014, pp. 9–16.

PERFORMANCE-PORTABLE BATCHED TENSOR CONTRACTIONS: LIBRARY DESIGN AND PERFORMANCE ANALYSIS

EDWARD HUTTER* AND SIVA RAJAMANICKAM†

Abstract. TCKK is a software package currently in development within the Kokkos Kernels library that supports performance-portable small, dense, batched tensor contractions for highly parallel MIC architectures and GPUs. Targeted applications include finite element codes that reformulate assembly steps into batched tensor contractions, and convolutional neural network kernels ubiquitous in machine learning. The goal of this project is to provide library support for performance-portable batched tensor contractions that rival leading implementations within the CEED and MAGMA frameworks. This report describes TCKK and will present its current interface, as well as early performance results on the Intel Knights Landing architecture.

1. Introduction. Tensor contractions have been present in a spectrum of research areas for many years, and with recent improved library support for distributed-memory, GPU, and heterogeneous architectures, an increasing number of applications are casting linear algebraic computation into tensor contractions. Libraries such as [7] and [9] focus on large dense tensor contractions on single-node architectures. More general libraries provide auto-generated code for both sparse and dense tensor contractions [6], while others offer performance portability for tensor contractions arising from BLAS-like primitives [8]. A few libraries have been recently developed to efficiently contract small tensors present in high-order finite element method simulations on GPUs [1, 10]. TCKK aims to differentiate itself by offering compiler-generated performance-portable small batched tensor contractions.

2. FEM tensor contractions. We explore three tensor contractions below. Tensor contraction #1 comes from the time term of a scalar advection diffusion problem in Nalu, a code for generalized unstructured massively parallel low Mach flow, where a nodal quadrature is used on a tensor-product hex element and W tensors represent integration operations. Tensor contractions #2 and #3 below are tensor formulations for the high-order finite element kernels of the Lagrangian phase of BLAST derived in [1]. These formulations solve the Euler equations of compressible hydrodynamics in a moving Lagrangian frame [2, 3].

1. $U_{d,l,m,n} = \sum_{i,j,k} W_{i,l} W_{j,m} W_{n,k} Z_{d,i,j,k}$
2. $U_{i_1,i_2} = \sum_{k_1,k_2,j_1,j_2} B_{k_1,i_1} B_{k_2,i_2} D_{k_1,k_2} B_{k_1,j_1} B_{k_2,j_2} V_{j_1,j_2}$
3. $U_{i_1,i_2,i_3} = \sum_{k_1,k_2,k_3,j_1,j_2,j_3} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} B_{k_1,j_1} B_{k_2,j_2} B_{k_3,j_3} V_{j_1,j_2,j_3}$

Naive implementations taken directly from the tensor contraction equations above yield computational costs of $\mathcal{O}(n^7)$, $\mathcal{O}(n^6)$, and $\mathcal{O}(n^9)$, respectively, where n is the length of all modes. Computational costs of $\mathcal{O}(n^5)$, $\mathcal{O}(n^3)$, and $\mathcal{O}(n^4)$, respectively, can be achieved by contracting individual tensors using matrix multiplications, for which highly-optimized routines exist. One such GEMM transformation is enumerated below for contraction #3, with GEMM calls highlighted in blue and non-GEMM computation highlighted in red. The enumeration of tensor contractions #1 and #2 follow closely from the enumeration below.

$$U = \sum_{k_1,k_2,k_3,j_1,j_2,j_3} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} B_{k_1,j_1} B_{k_2,j_2} B_{k_3,j_3} V_{j_1,j_2,j_3}$$

*University of Illinois at Urbana-Champaign, hutter2@illinois.edu
†Sandia National Laboratories, srajama@sandia.gov

$$
\begin{aligned}
&= \sum_{k_1,k_2,k_3,j_1,j_2} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} B_{k_1,j_1} B_{k_2,j_2} \left(\sum_{j_3} B_{k_3,j_3} V^T_{j_3,(j_1,j_2)} \right) \\
&= \sum_{k_1,k_2,k_3,j_1} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} B_{k_1,j_1} \left(\sum_{j_2} B_{k_2,j_2} Z_{k_3,(j_1,j_2)} \right) \\
&= \sum_{k_1,k_2,k_3,j_1} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} B_{k_1,j_1} \left(\sum_{j_2} B_{k_2,j_2} Z^T_{j_2,(k_3,j_1)} \right) \\
&= \sum_{k_1,k_2,k_3} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} \left(\sum_{j_1} B_{k_1,j_1} Z_{k_2,(k_3,j_1)} \right) \\
&= \sum_{k_1,k_2,k_3} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} \left(\sum_{j_1} B_{k_1,j_1} Z^T_{j_1,(k_2,k_3)} \right) \\
&= \sum_{k_1,k_2,k_3} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} D_{k_1,k_2,k_3} Z_{k_1,(k_2,k_3)} \\
&= \sum_{k_1,k_2,k_3} B_{k_1,i_1} B_{k_2,i_2} B_{k_3,i_3} E_{k_1,k_2,k_3} \\
&= \sum_{k_1,k_2} B_{k_1,i_1} B_{k_2,i_2} \left(\sum_{k_3} B_{k_3,i_3} E_{k_1,k_2,k_3} \right) \\
&= \sum_{k_1,k_2} B_{k_1,i_1} B_{k_2,i_2} \left(\sum_{k_3} B^T_{i_3,k_3} E^T_{k_3,(k_1,k_2)} \right) \\
&= \sum_{k_1,k_2} B_{k_1,i_1} B_{k_2,i_2} Z_{i_3,(k_1,k_2)} \\
&= \sum_{k_1} B_{k_1,i_1} \left(\sum_{k_2} B^T_{i_2,k_2} Z^T_{k_2,(i_3,k_1)} \right) \\
&= \sum_{k_1} B_{k_1,i_1} Z_{i_2,(i_3,k_1)} \\
&= \sum_{k_1} B^T_{i_1,k_1} Z^T_{k_1,(i_2,i_3)} \\
&= Z_{i_1,i_2,i_3}
\end{aligned}
$$

TCKK will aim to identify these transformations at compile-time by analyzing tensors represented as types. We explore this approach and detail TCKK's interface below.

3. TCKK design.

3.1. Overview. TCKK's performance-portable design is dependent on Kokkos [4], a C++ library aimed at providing performance-portability across multicore and GPU architectures. TCKK makes liberal use of three lightweight variadic class templates: *Modes*, *Indices*, and *Types*.

```
template<char...>
```

```
class Modes {};
```

```
template<size_t...>
class Indices {};
```

```
template<typename...>
class Types {};
```

A tensor's structure is completely described using *Modes*. For example, in TCKK, a tensor contraction $C_{m,n,p,q,r} = A_{n,k,q}B_{r,m,p,k}$ is described by the following types: Modes<'m','n','p','q','r'>, Modes<'n','k','q'>, and Modes<'r','m','p','k'>. The Indices class template allows variadic access to Kokkos::Views by faciliating pack expansions.

A number of lightweight classes and constexpr functions exist to manipulate these basic types at compile-time. Definitions will not be provided here.

3.2. Tensor allocation. TCKK constructs tensors as Kokkos::Views in either thread-local scratch memory, team-local scratch memory, or global memory, each encoded as types under the policy *MemoryTraits*. The *Tensor* class template detailed below is specialized only on these choices, regardless of execution space.

The internal datatypes of TCKK's tensors are complicated and thus hidden from the interface. These datatypes depend on the execution space (Kokkos::Cuda or Kokkos::OpenMP) because there exists a dedicated datatype within Kokkos for SIMD vectorization on MIC architectures. This datatype, a class template KokkosBatched::Experimental::Vector, was developed to vectorize small batched matrix multiplication with dimensions smaller than the vector lane width [5]. This datatype is exploited in similar spirit for small batched tensor contractions. The absense of such a datatype for the Kokkos::Cuda execution space motivates TCKK's choice to insert an extra mode of lengh 32 into the Kokkos::View. In either case, the fastest logical mode in the batched tensor data structure cycles through data belonging to unique tensors; each TCKK Tensor never holds a single tensor's data and can thus be called a tensor pack.

The Tensor class template below utilizes a typetrait *ModeType* internally to allocate a Kokkos::View object. A variadic template parameter pack, *indices*, containing an enumerated list of tuple indices, is used to expand the tuple *ModeLengths* into the Kokkos::View constructor. *NumBatches* is also provided as an argument to allow the allocation of multiple tensor packs into a single data structure. Note that *NumBatches* accounts for the slowest mode in the tensor object. The declarations for the three partial specializations are provided below.

```
// Tensor host class template
template<typename ScalarType,typename ExecutionSpace,
        typename MemoryTraits> class Tensor;

template<typename ScalarType,typename ExecutionSpace>
class Tensor<ScalarType,ExecutionSpace,GlobalMemory>{
public:
  template<typename... modetypes,size_t... indices>
  static auto create(std::tuple<modetypes...>& ModeLengths,
                     Indices<indices...>);

  template<typename... modetypes, size_t... indices>
```

```
  static auto create(std::tuple<modetypes...>& ModeLengths,
                     Indices<indices...>,size_t NumBatches);
};

template<typename ScalarType,typename ExecutionSpace>
class Tensor<ScalarType,ExecutionSpace,TeamScratchMemory>{
public:
  template<typename... modetypes,size_t... indices,
           typename TeamMemberType>
  KOKKOS_FUNCTION
  static auto create(const TeamMemberType& TeamInfo,
                     std::tuple<modetypes...>& ModeLengths,
                     Indices<indices...>, size_t MemoryLevel=0);
};

template<typename ScalarType,typename ExecutionSpace>
class Tensor<ScalarType,ExecutionSpace,ThreadScratchMemory>{
public:
  template<typename... modetypes,size_t... indices,
           typename TeamMemberType>
  KOKKOS_FUNCTION
  static auto create(const TeamMemberType& TeamInfo,
                     std::tuple<modetypes...>& ModeLengths,
                     Indices<indices...>, size_t MemoryLevel=0);
};
```

3.3. Tensor fill. To construct tensors directly in scratch memory, TCKK relies on the ability to fill tensors with data within a Kokkos::TeamPolicy. The *TensorFill* policy implemented by policy class *TensorCreateRandom* below requires filling a single pack of tensors with data. Any such policy class can be passed into the *Contract* class template as a template template parameter, to be used at the library's discretion. The values taken by parameters *NumThreads* and *ThreadID* are dependent on the pool of memory in which the tensors were allocated.

Nontype template parameter pack, *indices*, expands tuples *IndexPack* and *IndexBoundsPack* to allow variadic access to Kokkos::Views. Each thread fills a contiguous portion of the tensor with data and updates its tuple *IndexPack* at each iteration.

It must be noted that passing a *TensorFill* policy class to TCKK is necessary only to support TCKK's future ability to allocate, fill, and contract slices of tensors on-the-fly in the closest memory level possible.

```
template<typename ScalarType,typename ExecutionSpace>
class TensorCreateRandom{
public:
  template<typename... types,size_t... indices,
           typename TeamMemberType,typename TensorPackType>
  KOKKOS_FUNCTION
  static void invoke(const TeamMemberType& TeamInfo,
                     TensorPackType TensorPack,
                     std::tuple<types...>& IndexPack,
                     std::tuple<types...>& IndexBoundsPack,
                     Indices<indices...>,size_t NumThreads,
                     size_t ThreadID);
```

```
};
```

3.4. Tensor contraction. The *Contract* class template configures a number of policy classes together to provide access to many different contraction routines through a single interface. The *GemmThreadType* policy takes template parameters for transposing matrices A and B, as well as an algorithm type specified by the following template parameter *GemmAlgType*. Template parameter *MemoryPool* decides whether to allocate tensors in thread-local or team-local scratch memory.

Each tensor contraction must be called from inside a Kokkos::TeamPolicy. The length of the template template parameter pack *TensorFills* must match the length of *Contract*'s function template's template parameter pack *TensorModes* and *ModeLengthTuples*, as each input tensor allocated by *Contract* must have a size and must get filled with data. Each type within the *TensorModes* pack describes the mode-ordering of a unique input tensor and each type within *ModeLengthTuples* gives the mode-lengths for each tensor.

Both the Contract class template's declaration and its function template's declaration are given below. The implementation is not given as it comes directly from the procedure enumerated in Section 2.

```
template<typename ScalarType,
        template<typename,typename,typename> class GemmThreadType,
        class GemmAlgType,class MemoryPool,typename ExecutionSpace,
        template<typename,typename> class... TensorFills>
class Contract{
public:
  template<typename TeamMemberType,typename Tout,
          typename... TensorModes, typename... ModeLengthTuples>
  KOKKOS_FUNCTION
  static void invoke(const TeamMemberType& TeamInfo,Tout TensorOutput,
                     TCKK::Types<TensorModes...>,
                     ModeLengthTuples&& ModeLengths,
                     std::size_t MemoryLevel);
};
```

3.5. Example usage. An example on how TCKK can be utilized for tensor contraction #3 is given below.

```
  // Partial specialization for tensor contraction #3 (MethodID==3)
  template<typename ScalarType,
          template<typename,typename,typename> class GemmThreadType,
          class GemmAlgType,class MemoryPool,size_t MemoryLevel,
          typename ExecutionSpace>
  void Test<3,ModeLength,ScalarType,GemmThreadType,GemmAlgType,
           MemoryPool,MemoryLevel,ExecutionSpace>::
    test(size_t NumContractions,size_t NumTeams,
         size_t NumThreadsPerTeam,size_t ModeLength){

    // Describe the tensor contraction as a type
    using TensorContractionType
      = TCKK::Types<TCKK::Modes<'a','b','c'>, TCKK::Modes<'j','a'>,
                    TCKK::Modes<'k','b'>, TCKK::Modes<'l','c'>,
                    TCKK::Modes<'j','k','l'>, TCKK::Modes<'j','m'>,
                    TCKK::Modes<'k','n'>, TCKK::Modes<'l','o'>,
```

```
                TCKK::Modes<'m','n','o'>>;
// Assuming equal-length mode-lengths known at compile time
auto ModeLength3D = std::make_tuple(ModeLength,ModeLength,
                                    ModeLength);
auto ModeLength2D = std::make_tuple(ModeLength,ModeLength);

size_t ScratchMemorySize
  = TCKK::GetScratchMemorySize<ExecutionSpace>(MemoryLevel);
size_t PerThreadScratchMemorySize
  = TCKK::GetThreadScratchMemorySize<ExecutionSpace>();

// Get number of batches along the critical path
size_t NumBatches=0;
size_t BatchSize=0;
size_t BatchDiv=0;
if (std::is_same<TCKK::ThreadScratchMemory,MemoryPool>::value){
  auto CriticalPathBreadth = NumTeams*NumThreadsPerTeam
    * TCKK::VectorPackSize<ScalarType,ExecutionSpace>::size;
  BatchSize = NumTeams*NumThreadsPerTeam;
  BatchDiv = 1;
  NumBatches = NumContractions / CriticalPathBreadth;}
else if (std::is_same<TCKK::TeamScratchMemory,MemoryPool>::value){
  auto CriticalPathBreadth = NumTeams
    * TCKK::VectorPackSize<ScalarType,ExecutionSpace>::size;
  BatchSize = NumTeams;
  BatchDiv = NumThreadsPerTeam;
  NumBatches = NumContractions / CriticalPathBreadth;}
else {Kokkos::abort("Wrong MemoryPool type");}

// Build the output tensor as a pack of tensors
auto VectorSize
  = TCKK::VectorPackSize<ScalarType,ExecutionSpace>::size;
auto OutputTensor = TCKK::Tensor<ScalarType,ExecutionSpace,
  TCKK::GlobalMemory>::create(ModeLength3D,TCKK::Indices<0,1,2>(),
  NumContractions/VectorSize);

Kokkos::parallel_for(Kokkos::TeamPolicy<ExecutionSpace>(
  NumTeams, NumThreadsPerTeam).set_scratch_size(MemoryLevel,
    Kokkos::PerTeam(ScratchMemorySize),
    Kokkos::PerThread(PerThreadScratchMemorySize)),KOKKOS_LAMBDA(
      const typename Kokkos::TeamPolicy<ExecutionSpace>
        ::member_type& TeamInfo){
  auto TeamSize = MemoryPool::GetTeamSize(TeamInfo);
  auto ThreadIndex = MemoryPool::GetThreadIndex(TeamInfo);
  auto ThreadOffset = (TeamInfo.league_rank()*TeamInfo.team_size()
    + TeamInfo.team_rank()) / BatchDiv;

  for (size_t NumIter=0; NumIter < NumBatches; NumIter++){
    // Extract subview of OutputTensor
    auto SliceID = ThreadOffset + BatchSize*NumIter;
    auto OutputTensorTile =
      TCKK::GetSubTensor<ExecutionSpace>
        ::invoke(TeamInfo,OutputTensor,
        std::make_tuple(Kokkos::Impl::ALL_t(),
```

```
            Kokkos::Impl::ALL_t(),Kokkos::Impl::ALL_t(),SliceID));
      MemoryPool::Barrier(TeamInfo);
      Contract<ScalarType,GemmThreadType,GemmAlgType,MemoryPool,
        ExecutionSpace,TensorCreateRandom, TensorCreateRandom,
        TensorCreateRandom,TensorCreateRandom,TensorCreateRandom,
        TensorCreateRandom,TensorCreateRandom,TensorCreateRandom>
          ::invoke(TeamInfo, OutputTensorTile,
                   TensorContractionType(),ModeLength2D,
                   ModeLength2D,ModeLength2D,
                   ModeLength2D,ModeLength2D,
                   ModeLength2D,ModeLength2D,
                   ModeLength2D,MemoryLevel);
    }
  });
```

3.6. Tensor contraction code generation. For a single interface to suffice, the class template *Contract* must analyze its function template's template parameter pack *TensorModes* at compile time. By comparing and manipulating each *Modes* type hidden inside *TensorModes*, a sequence of GEMM calls and non-GEMM computational kernels can be identified. The enumerated steps given in Section 2 for tensor contraction #3 lead to a recursive strategy, with each step mapping to a single dedicated computation. This idea is currently in development.

4. Performance results. We hardcoded tensor contractions #1, #2, and #3 to test the performance of small, batched tensor contractions using the various GEMM routines that exist within Kokkos-Kernels. Each test performed 131072 individual tensor contractions with *ScalarType*=double. The number of teams was kept constant at 64, while the number of threads/team was varied among 1, 2, and 4. All tests were performed on an Intel Knights Landing architecture inside Sandia's Bowman testbed cluster, with the Kokkos::OpenMP execution space.

The performance results below include all work inside the Kokkos::TeamPolicy, including launching the TeamPolicy, allocating each input tensor, repeatedly filling the input tensor packs for each contraction, and the contraction itself. Table 4.1 enumerates the different policy classes passed into *Contract*. Columns B and C name classes and class templates present inside the KokkosBatched::Experimental namespace.

	A	B	C
0	Kokkos memory level 0	SerialGemm	Algo::Gemm::CompactMKL
1	Kokkos memory level 1	TeamGemm	Algo::Gemm::Unblocked
2	x	x	Algo::Gemm::Blocked

Table 4.1: Information table for the plots below

4.1. Tensor contraction #1. In each of Figures 4.1, 4.2, and 4.3, *MKL Compact* is the only GEMM routine showing significant performance divergence between memory levels 0 and 1. Suprisingly, the results indicate that contracting tensors allocated in high-bandwidth memory using MKLCompact achieve higher performance than contracting tensors allocated directly in cache. *MKL Compact*'s performance for tensors allocated directly in cache obtains the worst overall performance among all variants.

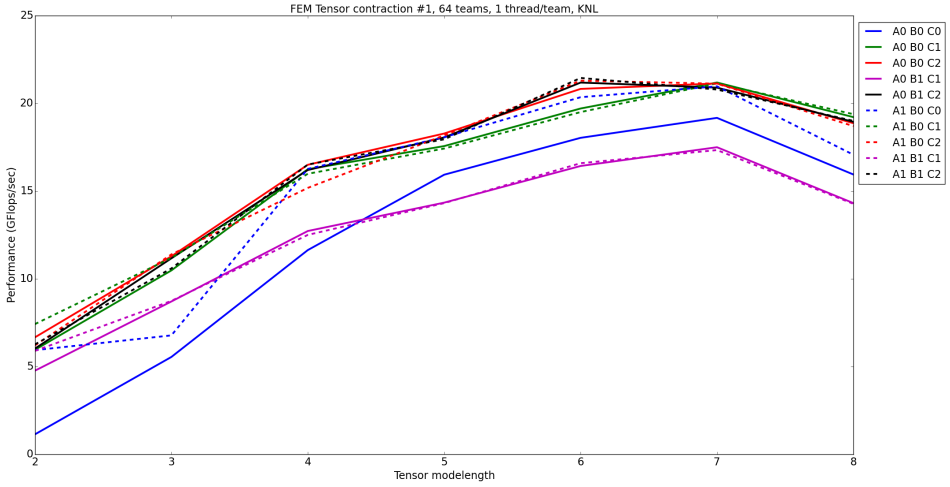


Fig. 4.1: Tensor contraction #1, 64 teams, 1 thread/team

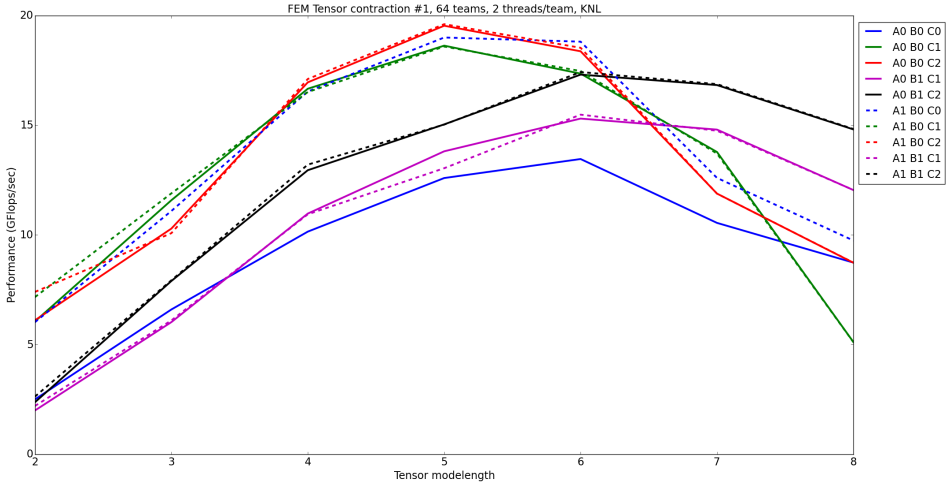


Fig. 4.2: Tensor contraction #1, 64 teams, 2 thread/team

Its evident that each GEMM variant scales poorly with the number of threads-per-team. In any *SerialGemm* method, such behavior can be explained by insufficient bandwidth among the hyperthreads for accessing L1 cache, and increased contention for other on-chip resources. Less contention seems to give both *TeamGemm* variants added performance at the larger mode-lengths, resulting in better scaling.

A potential performance pitfall shared by all variants is the increasingly non-square matrix multiplication that arises in tensor contraction #1. Note that a mode-length of

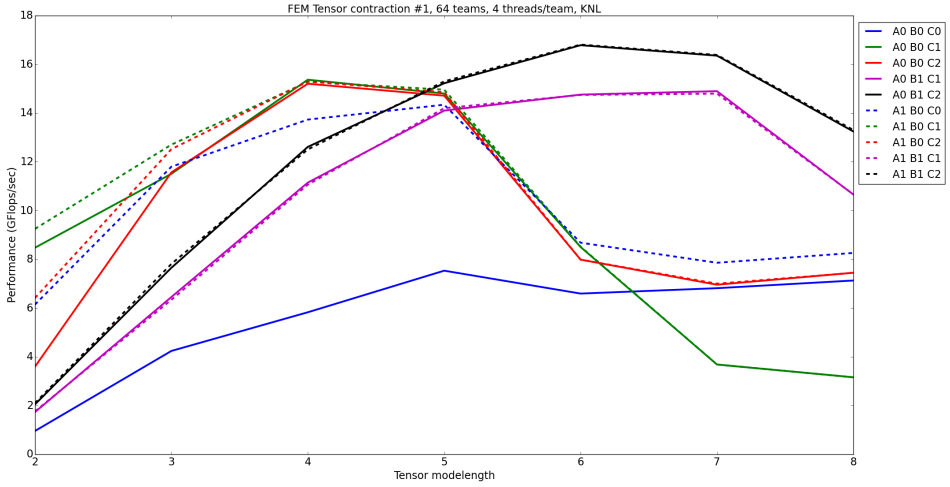


Fig. 4.3: Tensor contraction #1, 64 teams, 4 threads/team

$n = 8$ yields GEMMs with 8×8 and 8×512 matrices. The design of the KokkosBatched infastructure is not meant for non-square GEMMs, so we expect batching these into many square GEMMs to yield better scaling.

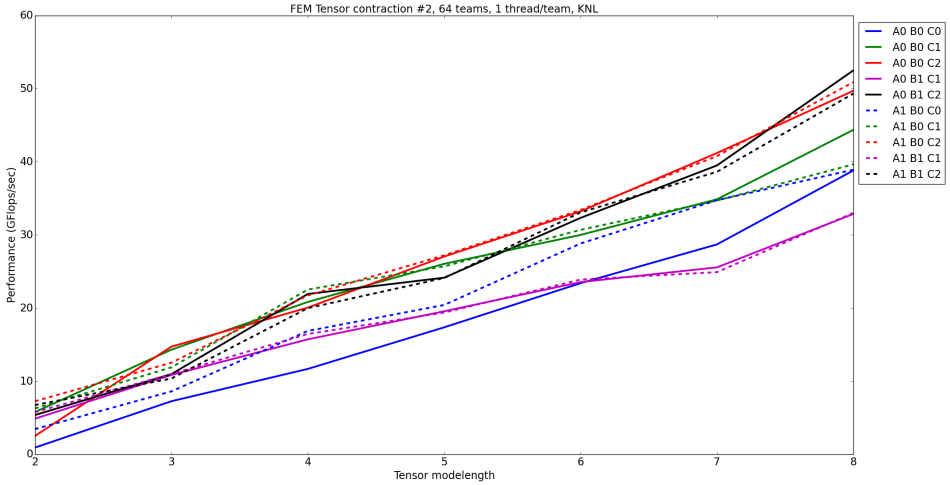


Fig. 4.4: Tensor contraction #2, 64 teams, 1 threads/team

4.2. Tensor contraction #2. The performance and scaling achieved by all variants for tensor contraction #2 in Figures 4.4, 4.5, and 4.6 far surpasses that achieved for tensor contraction #1. This can be attributed to the fact that only square GEMMs are present. As

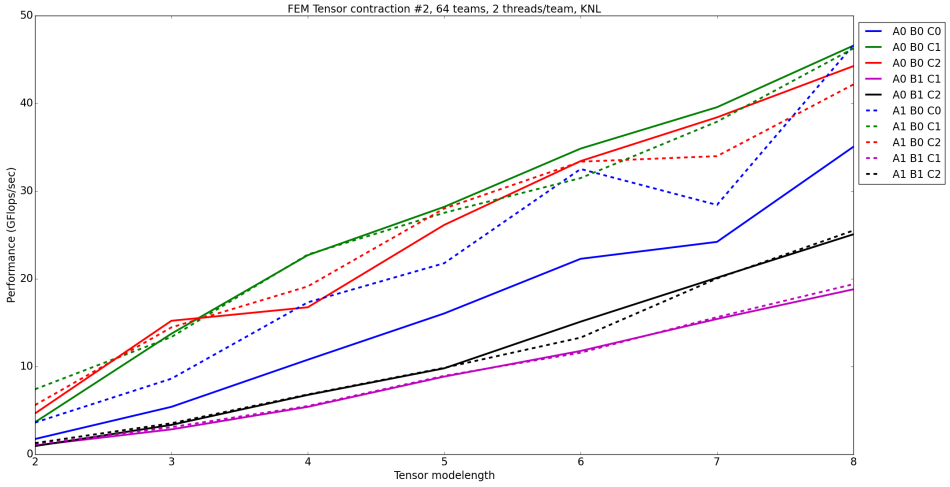


Fig. 4.5: Tensor contraction #2, 64 teams, 2 threads/team

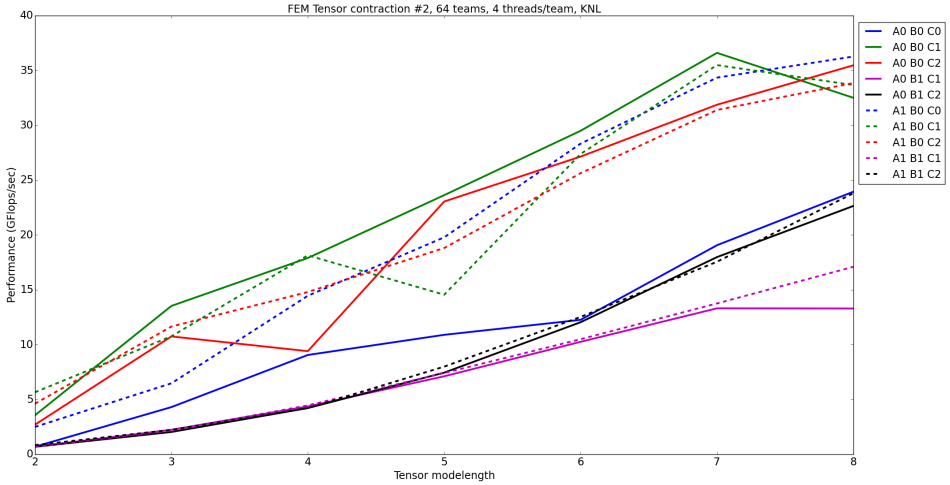


Fig. 4.6: Tensor contraction #2, 64 teams, 4 threads/team

before, the performance gap between memory levels is present only with the *MKL Compact* variant. The *TeamGemm* variants scale worse than the *SerialGemm* variants because there is insufficient work present in contracting a single pack of tensors of mode-length ≤ 8.

4.3. Tensor contraction #3. Tensor contraction #3 involves non-square GEMMs, yet not quite as large as those present in tensor contraction #1. As such, the Blocked *TeamGemm* scales the best with the higher thread counts, as sufficient work is available for each team to efficiently utilize multiple threads. The overall performance

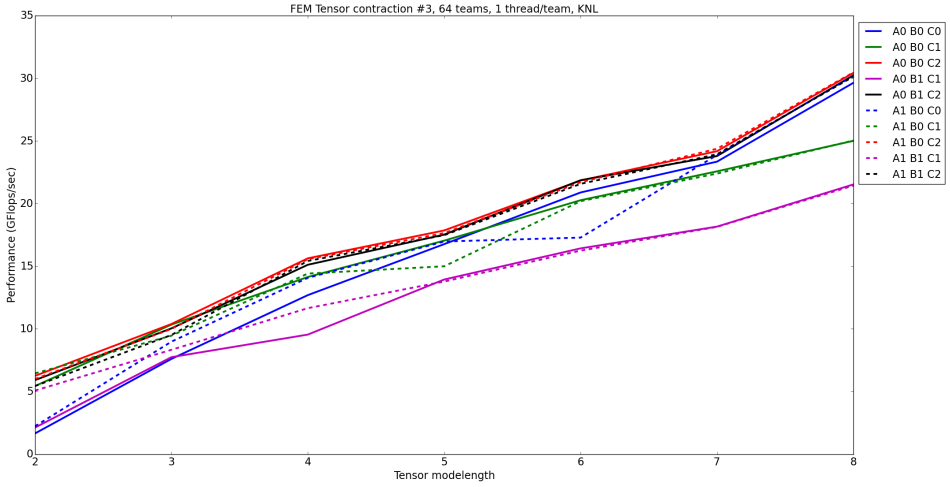


Fig. 4.7: Tensor contraction #3, 64 teams, 1 threads/team

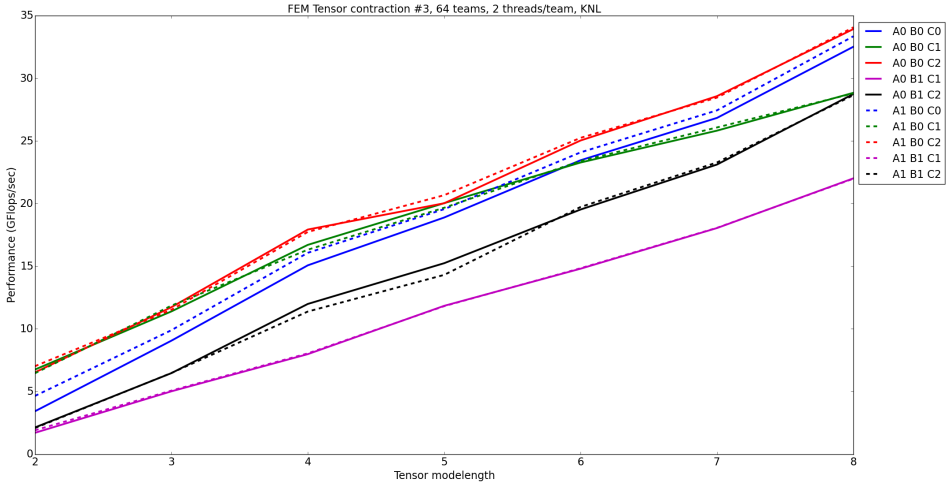


Fig. 4.8: Tensor contraction #3, 64 teams, 2 threads/team

attained in Figures 4.7, 4.8, and 4.9 is slightly lower than that achieved for tensor contraction #2, motivating the batching of the short-and-fat folded tensor into square GEMMs. This performance issue can also be attributed to the asymptotically larger number of bandwidth-bound operations that are currently being performed, such as folding, unfolding, and filling each input tensor.

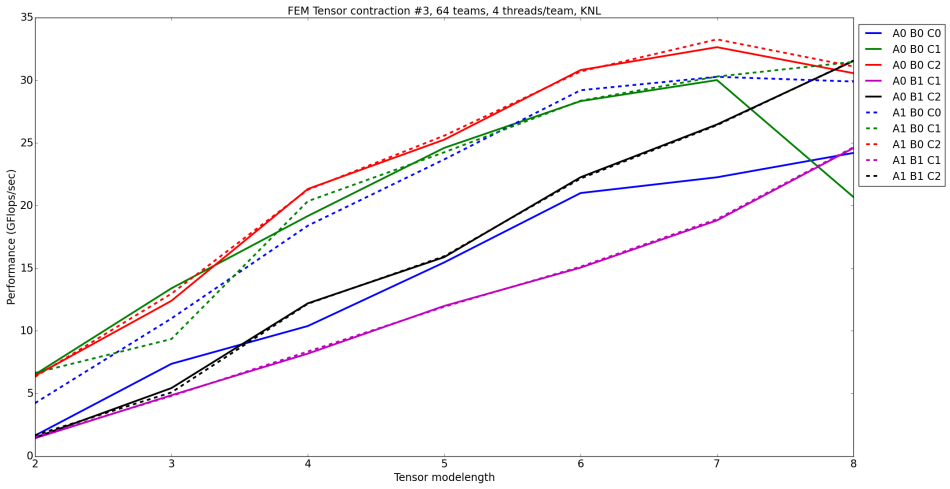


Fig. 4.9: Tensor contraction #3, 64 teams, 4 threads/team

5. Conclusions. We have introduced early performance results and the future interface of TCKK, a performance-portable library that performs small, batched tensor contractions using Kokkos. Its main contribution will be its ability to deduce a recursive sequence of computational steps at compile-time that can be handled in a performance-portable way by GEMMs and other computational kernels.

There remains numerous directions for future development of performance-portable tensor contractions present in finite element and convolutional neural-net codes. First, the current implementation can be greatly improved. The performance of contracting the smallest tensors directly built in the closest memory level (Kokkos level 0) is suprisingly worse than that of contracting tensors built in high-bandwidth memory (Kokkos level 1). Further, at a certain tensor mode-length, the performance difference is negligable. An investigation into this performance as well as into understanding the conditions under which Kokkos' allocations spill out to larger memory levels will influence whether further tensor slicing is required to allocate and contract solely in the closest memory level. Tensor slicing alone will require either knowledge of the mode-lengths at compile-time, or a JIT-like compilation.

The performance results of tensor contraction #1 motivate future work on compile-time recognition of tensor slicing, where slicing can avoid large, innefficient non-square GEMMs induced by folding. Tensor slicing will force changes to the *TensorFill* policy, as TCKK will need to fill tensor slices instead of entire tensors.

TCKK's support for tensor contractions on GPUs is not mature enough to warrant discussion in this report. The performance of various GEMM implementations in the KokkosBatched::Experimental namespace must be investigated.

Perhaps the largest avenue for future research involves compile-time code generaton for tensor contractions of any flavor. Compile-time infastructure is already in place for analyzing the typelists that represent each tensor. We seek optimal slicing, folding, and contracting strategies using tensor structure available at compile-time. Such a design strategy fits the template metaprogramming approach utilized by Kokkos.

REFERENCES

[1] A. Abdelfattah, M. Baboulin, V. Dobrev, J. J. Dongarra, C. W. Earl, J. Falcou, A. Haidar, I. Karlin, T. V. Kolev, I. Masliah, and S. Tomov, *High-performance tensor contractions for gpus*, in ICCS, vol. 80 of Procedia Computer Science, Elsevier, 2016, pp. 108–118.

[2] V. Dobrev, T. V. Kolev, and R. N. Rieben, *High-order curvilinear finite element methods for lagrangian hydrodynamics*, SIAM J. Scientific Computing, 34 (2012).

[3] T. Dong, V. Dobrev, T. Kolev, R. Rieben, S. Tomov, and J. Dongarra, *A step towards energy efficient computing: Redesigning a hydrodynamic application on cpu-gpu*, in 2014 IEEE 28th International Parallel and Distributed Processing Symposium, May 2014, pp. 972–981.

[4] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[5] K. Kim, T. B. Costa, M. Deveci, A. M. Bradley, S. D. Hammond, M. E. Guney, S. Knepper, S. Story, and S. Rajamanickam, *Designing vector-friendly compact blas and lapack kernels*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, New York, NY, USA, 2017, ACM, pp. 55:1–55:12.

[6] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, *The tensor algebra compiler*, Proc. ACM Program. Lang., 1 (2017), pp. 77:1–77:29.

[7] D. Matthews, *High-performance tensor contraction without BLAS*, CoRR, abs/1607.00291 (2016).

[8] Y. Shi, U. N. Niranjan, A. Anandkumar, and C. Cecka, *Tensor contractions with extended BLAS kernels on CPU and GPU*, CoRR, abs/1606.05696 (2016).

[9] P. Springer and P. Bientinesi, *Design of a high-performance gemm-like tensor–tensor multiplication*, ACM Trans. Math. Softw., 44 (2018), pp. 28:1–28:29.

[10] K. Swirydowicz, N. Chalmers, A. Karakus, and T. Warburton, *Acceleration of tensor-product operations for high-order finite element methods*, CoRR, abs/1711.00903 (2017).

FAST TRIANGLE COUNTING USING CILK

ABDURRAHMAN YAŞAR*, SIVASANKARAN RAJAMANICKAM†, MICHAEL WOLF‡, JONATHAN BERRY§, AND ÜMIT V. ÇATALYÜREK¶

Abstract. Triangle counting is a representative graph analysis algorithm with several applications. It is also one of the three benchmarks used in the IEEE HPEC Graph Challenge. Triangle counting can be expressed as a graph algorithm and in a linear algebra formulation using sparse matrix-matrix multiplication (SpGEMM). The linear algebra formulation using the SpGEMM algorithm in the Kokkos kernels library was one of the fastest implementations for the triangle counting problem in last year's Graph Challenge. This paper improves upon that work by developing an SpGEMM implementation that relies on a highly efficient, work-stealing, multithreaded runtime. We demonstrate that this new implementation results in improving the triangle counting runtime up to 5× to 12× on different architectures. This new implementation also breaks the 10^9 barrier for the rate measure on a single node for the triangle counting problem. We also compare the linear algebra formulation with a traditional graph based formulation. The linear algebra implementation is up to 2.96× to 7× faster on different architectures compared to the graph based implementation. Furthermore, we present analysis of the scaling of the triangle counting implementation as the graph sizes increase using both synthetic and real graphs from the graph challenge data set. This report is firstly submitted to Graph Challenge 2018 [29].

1. Introduction. Triangle counting is an important kernel for graph analysis and forms the core of several important graph algorithms such as triangle enumeration, subgraph isomorphism, dense neighborhood graph discovery [27], and link recommendation [25]. Triangle counting is one of the three IEEE HPEC Graph Challenge problems [19] and is important for k-truss computation, another Graph Challenge problem [11]. The 2017 graph challenge had several papers related to the triangle counting problem improving the state-of-the-art for a foundational graph analysis kernel (see Section 2 for a summary). Samsi et al. [18] analyzed the different results presented at the workshop to understand the state-of-the-art for the triangle counting problem. Typical results for the fastest submissions in 2017 Graph Challenge were on the order of 10^8 edges per second for the triangle counting problem.

The primary motivation of this paper is to improve the state-of-the-art for triangle counting, using an efficient, task-stealing, multithreaded runtime – Cilk [4]. This paper improves the earlier linear algebra based triangle counting implementation KKTri, which was one of the "champions" of the 2017 Graph Challenge workshop [28], by replacing OpenMP with Cilk. The choice of new runtime was motivated by an analysis of the OpenMP implementation's scalability for larger problems and large number of threads. Section 3 describes the algorithms that are implemented and the Cilk based implementation. The differences between these two runtimes for the triangle counting problem on irregular graphs are evident based on the results (see Section 4).

The primary contribution of this paper is a Cilk based implementation (KKTri-Cilk) of the KK-SpGEMM algorithm from the Kokkos kernels library [14] and a Cilk-based triangle counting implementation using the new SpGEMM. The focus is on obtaining the best performance on a single multicore node. Our triangle counting implementation surpasses 10^9 for the rate measure on a single node for the first time. Previous work has reported such rates, but only when ignoring preprocessing steps in the calculated runtime (e.g., [3]). Furthermore, detailed experiments comparing an OpenMP (KKTri) and Cilk (KKTri-Cilk)

*Georgia Institute of Technology, Atlanta, GA, U.S.A.
†Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, U.S.A.
‡Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, U.S.A.
§Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, U.S.A.
¶Georgia Institute of Technology, Atlanta, GA, U.S.A.

based triangle counting demonstrate the Cilk based triangle counting results in a speedup of up to 5× to 12× on different architectures. KKTri-Cilk is up to 2.96× to 7× faster on different architectures when compared with the graph based implementation (TCM) [21]. Another significant contribution is a scalability analysis of triangle counting for synthetic and real graphs. This analysis shows that the runtime per vertex for triangle counting is highly correlated with $\frac{4}{3}$-moment of the graph. Finally, we show the correlation between the rates and the conductance of graph.

2. Background.

2.1. KKTri and 2017 Static Graph Challenge. For the 2017 Static Graph Challenge [19], we submitted a linear algebra-based triangle counting implementation KKTri (previously designated TCKK) [28], focused on efficient single node parallelism. KKTri was built upon the performance portable SpGEMM (called KK-MEM) [9] in the Kokkos kernels library [14].

We described two linear-algebra based formulations of triangle counting that were based on the adjacency matrix of the graph [19]. The first formulation was a slight variant of Azad, et al. [1] (a linear algebra-based formulation of Cohen's algorithm [6]). This formulation represented triangle counting in terms of sparse matrix-matrix multiplication followed by an element-wise matrix multiplication: $D = (L \cdot U) . * L$, where L and U are the lower and upper triangle parts of the adjacency matrix for the graph, respectively (Azad et al. used A for the element-wise multiply). The SpGEMM operation can be fused with the the element-wise multiply by using the rightmost L as a mask for the SpGEMM, which avoids explicitly storing all the non-zeros resulting from $L \cdot U$. However, even with this optimization, each triangle is "counted" twice (with one being filtered out). This method was not evaluated in the previous work. Instead, we presented a second formulation, $D = (L \cdot L) . * L$, which we used for the 2017 Graph Challenge. This formulation follows the same logic as the previous method with the SpGEMM operation counting wedges and element-wise multiplication filtering out the wedges that are not triangles. However, this formulation resulted in an additional constraint on the wedges "visited", which reduced the number of wedges stored after the SpGEMM operation. Typically, we saw a reduction in the number of operations and runtime. Both formulations are used in this paper and implemented using Cilk.

Previously, we found three optimizations that helped in achieving good performance. First, masked SpGEMM (mentioned before) reduced the memory needed for triangle counting. Secondly, our masked SpGEMM operation used data compression on the right hand side matrix. Finally, the ordering of the graph before forming the lower and upper triangle matrices is essential for good performance. All three optimizations are used in the Cilk implementation described here as well.

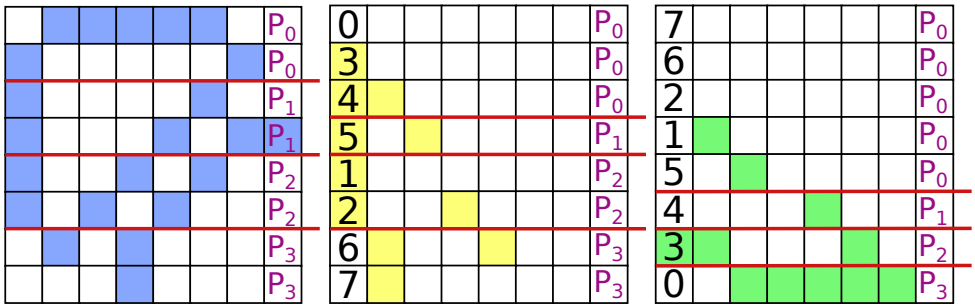
KKTri was one of the fastest triangle counting methods in the 2017 Static Graph Challenge with a top rate (defined to be the number of undirected edges in the graph divided by the runtime) of 637 million edges per second on a single multicore system. We also compared KKTri to the Cilk implementation of parallel merge-based triangle counted method (TCM) [21] (TCM-Cilk). KKTri performed favorably to TCM-Cilk except when compression did not help or when hyperthreads were used. This observation led us to consider developing a Cilk based implementation of our algorithm, the focus of this paper.

The KK-MEM algorithm in Kokkos kernels library has been improved with a meta-algorithm KK-SpGEMM that chooses between different accumulators, compression scheme and hashmaps [10]. This will be the basis of the Cilk implementation as well. The meta-algorithm has been improved further to support Cilk runtime specific optimizations.

The 2017 Graph Challenge has spurred advancements in the state-of-the-art triangle counting. These include advancements in shared-memory parallel [13, 22, 24], single node

accelerator [3], multiple accelerator [12], operator-based [26], and distributed-memory [17] implementations. We also compare favorably to the Graph BLAS numbers in SuiteSparse [7] on the IBM Power8 architecture using 160 threads (e.g. 63% faster on friendster using 96 threads).

3. Approach.



(a) Matrix Form (b) LL Alg: L Matrix (c) LU Alg: L Matrix

Fig. 3.1: Example problem. (a) is the matrix representation of a graph, each colored cell represents an edge. (b) and (c) are the lower triangular matrices of (a) ordered for LL and LU algorithms, respectively. In (a-c) permutation and partition ids of the rows are written in the first and last column, respectively. Red line is the partition border.

3.1. Cilk Based KKTri Algorithm. KKTri-Cilk algorithm inherits from the KK-SpGEMM algorithm described in [10] and tries to improve load-balancing and efficient hyper-thread usage issues using Cilk based programming model and optimizations. The Cilk programming model supports different parallel patterns such as `cilk_for`, and `cilk_spawn`. Cilk tasks are lightweight with efficient task-stealing allowing an implicit way to load balance. Aside from the runtime system, the parallelization strategy is the main difference between KKTri-Cilk and KKTri. KKTri used a very simple scheme, partitioning the matrix evenly into partitions of a fixed number of rows. KKTri-Cilk also creates row-wise partitions (with each partition being assigned to a Cilk task) but not with a constant number of rows per partition. Instead, a KKTri-Cilk partition P_i stores its border in two pointers, one for its first row and one for its last row.

To balance the work among the tasks, KKTri-Cilk uses an heuristic to find the partitions, creating partitions such that the number of non-zeros within each partition are approximately equal. This is a greedy block (row) partitioning approach to balance the number of non-zeros in a partition. For instance, in our example (Fig. 3.1b), we have four partitions (divided by the red lines), $P = \{P_0, P_1, P_2, P_3\}$, with these partitions have $3, 2, 3, 3$ non-zeros respectively. The orderings used for triangle counting help reduce the number of dense rows and achieve better partitions as well. We partition the L matrix for both LL and LU algorithms.

Algorithm 1 outlines the basic KKTri-Cilk triangle counting algorithm. Each task is responsible for counting the number of triangles in its partition, P_i. These partitions are disjoint therefore tasks can be freely executed in parallel. Since each partition may contain different numbers of rows, `cilk_for` is not used. Instead, `cilk_spawn` is used to initiate these tasks and execute them in parallel. Experimentally, we saw that this approach gave better performance than using `cilk_for` with different grain sizes. Finally, when each task has finished computing the number of triangles within its partition (ensured by the synchronize

Algorithm 1 KKTRI-CILK(G)

Graph G is given in CSR format
$n_{|T|} \leftarrow 0 \Rightarrow$ Initialize number of triangles.

Decide algorithm then order and set A - B matrices
$\langle A, B \rangle \leftarrow$ ANALYZEMATRIX(G)

PARTITIONS ARE BALANCED BASED ON NUMBER OF NON-ZEROS.
$P \leftarrow$ CREATEPARTITIONS(G)

RUN KKTRI ALGORITHM IN PARALLEL FOR EACH P_i
for each $P_i \in P$ **do**
    `cilk_spawn` KKTRISPGEMM-CILK($A, B, P_i, n_{|T|}$)
    `cilk_sync`
end for
return $n_{|T|}$

Algorithm 2 KKTRISPGEMM-CILK($A, B, P_x, n_{|T|}$)

A is lower triangular matrix.
B is lower or upper triangular matrix; depends on the alg.
P_i is the partition, keeps first and last row's pointers.
$n_{|T|}$ is shared between tasks.
$n_t \leftarrow 0 \Rightarrow$ Count of the local number of triangles
$H \leftarrow \varnothing \Rightarrow$ Initialize hashmap accumulator
for each $i \in P_x$ **do**
    $H[v \in B(i)] \leftarrow i$
    for each $j \in A(i)$ **do**
        for each $k \in B(j)$ **do**
            $n_t \leftarrow n_t + 1$ **if** $H[k] = i$ **else** 0
        end for
        $H \leftarrow \varnothing \Rightarrow$ Clean hashmap
    end for
    Atomically add local number of triangles.
end for
ATOMICADD($n_{|T|}, n_t$)

step), the global triangle count has been atomically updated to give the final answer.

Each task does a matrix multiplication on its partition. As described in the background, matrix multiplication based triangle counting can be solved using two different algorithms; LL and LU. Algorithm 2 describes these methods. A represents the lower triangular matrix in both algorithms. B is the upper (lower) triangular matrix in LU (LL) algorithm. In both algorithms, we multiply A and B matrices, $C = A \cdot B$, and then mask their result, C, with lower triangular matrix ($C. * L$). KKTri-Cilk implements an in-place masking strategy to reduce number of operations and also memory movement. For a given row, $i \in P_x$, all non-zeros , $v \in B(i)$, are inserted into a hashmap, H, by using their column ids as the key and i as their value. For each non-zero column, j, of $A(i)$ we visit $B(j)$ and check H. If a non-zero column $k \in B(j)$ is set to i, we do in-place masking. If $H[k]$ is equal to i, we have found a triangle. The hashmap needs to be reset for each row in P_x. Finally, we add the local triangle count to the global one using atomic addition.

KKTri-Cilk inherits some of the techniques, and the data structures such as compression,

linked-list based hashmap accumulator and dense hashmap accumulator from KKTri [10] and optimizes them for the Cilk implementation.

Hashmap accumulator is critical role for the efficiency of the KKTri algorithm. As previously defined data structures for this problem are highly optimized, we use KKTri's linked list based and dense hash accumulators with minor changes. First, in KKTri-Cilk a graph is partitioned into disjoint sets and, each of these disjoint sets are executed in parallel. Therefore, there are no concurrent inserts, which would require atomic compare and swap instructions. KKTri-Cilk simply defines a hashmap accumulator for each parallel task. Second, as shown in Alg. 2 when execution of a row is completed, we need to reset the hashmap. This can be done in two ways; first, all entries can be reset, second used hashes can be tracked and cleaned. For, the first case using Cilk's array notation to reset all of the elements in a vectorized fashion gives better performance. If the number of used hashes are less than the 50%, cleaning the tracked hashes results in better performance.

Compression of the right hand side matrix can decrease the problem size significantly, and allow using efficient bitwise operations. However, compression is not always successful because of the natural order of the matrices. If there is low locality in the matrix (e.g. RMAT graphs) then compression doesn't improve the execution time [28]. Locality of the partitions which are being processed by tasks has an important effect on efficient usage of the memory bandwidth and the cache. Hence, locality can significantly impact the triangle counting rate obtained by KKTri. In this work, we use conductance as as a way to evaluate locality for a given matrix. In this context, conductance is defined as the ratio between the number of non-zeros in a partition where rows of their column indices appear in different partitions, and total number of non-zeros within that partition. Denoted by C^d, formally conductance of a partition, P, can be defined as follows:

$$C^d(P) = \frac{|\{A(u,v) \neq 0 : |\{u,v\} \cap P| = 1\}|}{|\{A(u,v) \neq 0 : |\{u,v\} \cap P| > 0\}|} \tag{3.1}$$

For example, in the matrix represented in Figure 3.1a, P_0 has 2 rows 7 non-zero elements, 2 of the non-zeros ($A(0,1)$ and $A(1,0)$) can be accessed within this partition. Therefore we can define conductance of this partition as follows; $C^d(P_0) = \frac{5}{7} = 0.71$. We use the average of the conductance of 16-way partition to study the locality of a graph (Table 4.2).

4. Experimental Evaluation. We present several experiments to identify the performance trade-offs of the KKTri-Cilk algorithm. These experiments were carried out on three architectures with multicore processors that are shown in Table 4.1. Intel compiler (`icpc`) version 18.1.163 is used to compile both KKTri and TCM codes. We use 2 hyperthreads for Skylake and Haswell architectures.

4.1. Dataset and Peak Rate. Table 4.2 lists 27 graphs that we used in our experiments along with the number of vertices ($|V|$), number of edges ($|E|$), number of triangles ($|T|$), complement of the conductance ($1 - C^d$), 4/3 moment. Table 4.3 lists execution time in seconds on Skylake server, and the rate between edges and execution time (Rate) on three different architectures. In addition to 20 Challenge graphs for which triangle computing is particularly costly (based on the reference implementation), 7 additional large real-world graphs [5,8,16] are included in our experiments. We used the Graph Challenge procedure of symmetrizing the matrices (using undirected graphs). For all experiments, we report the median time of five runs. Table 4.3 shows the best achievable rate of 1.14×10^9 for the uk-2005 matrix. We are also able to achieve more than 10^9 rate for wb-edu matrix. This is about 2× better the reported peak rate in the 2017 Graph Challenge. All three graphs have very high locality (based on their conductance). (In general, we observe a high

Table 4.1: Characteristics of the architectures used.

	Server 1	**Server 2**	**Server 3**
Code Name	Skylake	Haswell	KNL
Model	Intel Xeon Platinum 8160	Intel Xeon E7-4850 v3	Intel Xeon Phi 7250
Cores/Freq.	$2 \times 24/2.1$GHz	$4 \times 14/2.2$GHz	$4 \times 68/1.4$GHz
Cache/Mem	33MB/196GB	35MB/16GB	1(MB)/16GB

correlation (0.91) between the conductance and the rate achieved.) The Skylake architecture results in the best runtimes for all graphs except friendster. Table 4.3 also has the times highlighted in green when KKTri-Cilk is the fastest method compared KK-OMP and the TCM implementations.

Table 4.2: Properties of the dataset. Conductance is reported for lower triangular matrices. Moment is the 4/3 Moment.

Data Set	$\|V\|$	$\|E\|$	$\|T\|$	$1 - C^d$	**Moment**
cit-HepTh	27,770	352,285	1,478,735	0.141	98.079
email-EuAll	265,214	364,481	267,313	0.112	13.521
soc-Epinions1	75,879	405,740	1,624,481	0.086	48.062
cit-HepPh	34,546	420,877	1,276,868	0.091	86.684
soc-Slashdot0811	77,360	469,180	551,724	0.067	50.726
soc-Slashdot0902	82,168	504,230	602,592	0.069	51.416
flickrEdges	105,938	2,316,948	107,987,357	0.098	268.872
amazon0312	400,727	2,349,869	3,686,467	0.229	30.455
amazon0505	410,236	2,439,437	3,951,063	0.233	31.100
amazon0601	403,394	2,443,408	3,986,507	0.276	17.434
scale18	174,147	3,800,348	82,287,285	0.059	347.232
scale19	335,318	7,729,675	186,288,972	0.058	395.145
as-Skitter	1,696,415	11,095,298	28,769,868	0.17	73.835
scale20	645,820	15,680,861	419,349,784	0.059	448.022
cit-Patents	3,774,768	16,518,947	7,515,023	0.027	21.888
scale21	1,243,072	31,731,650	935,100,883	0.059	506.130
soc-LiveJournal1	4,847,571	42,851,237	285,730,264	0.242	53.766
wb-edu	9,845,725	46,236,105	254,718,147	0.938	16.775
scale22	2,393,285	64,097,004	2,067,392,370	0.058	569.872
scale23	4,606,314	129,250,705	4,549,133,002	0.059	640.093
scale24	8,860,450	260,261,843	9,936,161,560	0.059	717.548
scale25	17,043,780	523,467,448	21,575,375,802	0.059	802.552
uk-2005	39,459,925	783,027,125	21,779,366,056	0.925	90.495
it-2004	41,291,594	1,027,474,947	48,374,551,054	0.942	125.144
twitter	61,578,414	1,202,513,046	34,824,916,864	0.126	687.587
friendster	65,608,366	1,806,067,135	4,173,724,142	0.182	344.160
uk-2007	105,896,555	3,301,876,564	286,701,284,103	0.968	144.436

4.2. Runtime Comparison. Table 4.4 presents observed speedups using the Cilk programming model compared to the OpenMP runtime for both KKTri and TCM algorithms, on three different architectures. The geometric mean of the speedups is in the last row of the table. In Table 4.4 each cell is highlighted with yellow if the Cilk implementation is faster than the OpenMP implementation. The KKTri-Cilk of the KKTri algorithm outperforms the KKTri-OpenMP in 63 of 78 (81%) cases. For TCM, these numbers are 70 out of 78 (90%) cases. Clearly, Cilk helps to getter better performance for both algorithms.

Although KKTri-Cilk outperforms KKTri-OpenMP in all of the problems on Haswell,

Table 4.3: Best of the medians of the execution times in seconds on Skylake is reported. Highlights: Green - KK-Cilk is better than TCM, Red - the fastest rate for a graph, Blue- the best rate among all graphs and the rate for the largest graph.

Data Set	Time (s)	Rates		
		Skylake	Haswell	KNL
cit-HepTh	0.003	1.20E+08	8.24E+07	1.54E+07
email-EuAll	0.003	1.16E+08	1.10E+08	2.16E+07
soc-Epinions1 0.004	1.06E+08	6.72E+07	2.44E+07	
cit-HepPh	0.004	1.11E+08	8.77E+07	2.47E+07
soc-Slashdot0811	0.004	1.18E+08	7.97E+07	2.71E+07
soc-Slashdot0902	0.003	1.57E+08	8.64E+07	2.77E+07
flickrEdges	0.013	1.85E+08	1.15E+08	2.99E+07
amazon0312	0.006	3.87E+08	2.51E+08	9.34E+07
amazon0505	0.006	3.79E+08	2.75E+08	9.36E+07
amazon0601	0.006	4.17E+08	2.87E+08	9.81E+07
scale18	0.031	1.24E+08	1.07E+08	2.88E+07
scale19	0.075	1.04E+08	8.06E+07	2.79E+07
as-Skitter	0.026	4.23E+08	3.23E+08	1.23E+08
scale20	0.184	8.53E+07	5.63E+07	2.50E+07
cit-Patents	0.028	5.82E+08	4.21E+08	1.22E+08
scale21	0.511	6.21E+07	4.78E+07	2.01E+07
soc-LiveJournal1	0.137	3.14E+08	2.28E+08	1.07E+08
wb-edu	0.042	1.10E+09	6.55E+08	1.48E+08
scale22	1.581	4.05E+07	3.50E+07	1.71E+07
scale23	3.786	3.41E+07	2.62E+07	1.45E+07
scale24	10.282	2.53E+07	2.04E+07	1.21E+07
scale25	25.652	2.04E+07	1.88E+07	9.11E+06
uk-2005	0.684	1.14E+09	9.35E+08	2.59E+08
it-2004	1.293	7.95E+08	5.86E+08	1.47E+08
twitter	28.359	4.24E+07	4.46E+07	N/A
friendster	18.552	9.74E+07	7.93E+07	N/A
uk-2007	3.545	9.31E+08	7.49E+08	N/A

the OpenMP version outperforms the Cilk version in smaller instances on Skylake. Since Skylake gives the best performance in almost all problems, we hypothesize this could be due to the tasking overhead in Cilk for these small graph instances. TCM's Cilk implementation outperforms the OpenMP implementation in all problems on both Haswell and Skylake architectures with a geometric mean of 2.6 and 2.3. Part of this difference could be attributed to optimizations used in different implementations. However, there is a clear trend that Cilk helps these algorithms on Haswell and Skylake architectures. However, in KNL architecture OpenMP implementations become competitive with Cilk implementations in many problems and geometric mean of the speedup decreases to 1.07× and 1.65× for KKTri and TCM. This could be due to the relatively smaller caches on the KNL architectures not helping the tasking runtime.

4.3. Strong Scaling. Figure 4.1 presents strong scaling speedup of KKTri and TCM for three graphs - Friendster, uk2005 and scale24 - on Skylake architecture. KKTri-Cilk scales the best in all three problems when scaled by the *best* sequential execution time among the different methods. For the uk-2005 and scale24 graphs, scaling is better before using hyperthreads. Hyperthreads give a small but measurable benefit. For Friendster, all implementations show better scaling with hyperthreads. Figure 4.1h shows the concentration of non-zeroes in the uk-2005 graph. This graph has a very good ordering and most of the non-zeros appear to be near diagonal resulting in highly local computations. Also, we observe from Figure 4.1e that KKTri algorithms' sequential execution time is also less than TCM thanks to these local computations. In contrast to the uk-2005 graph, the scale24 graph

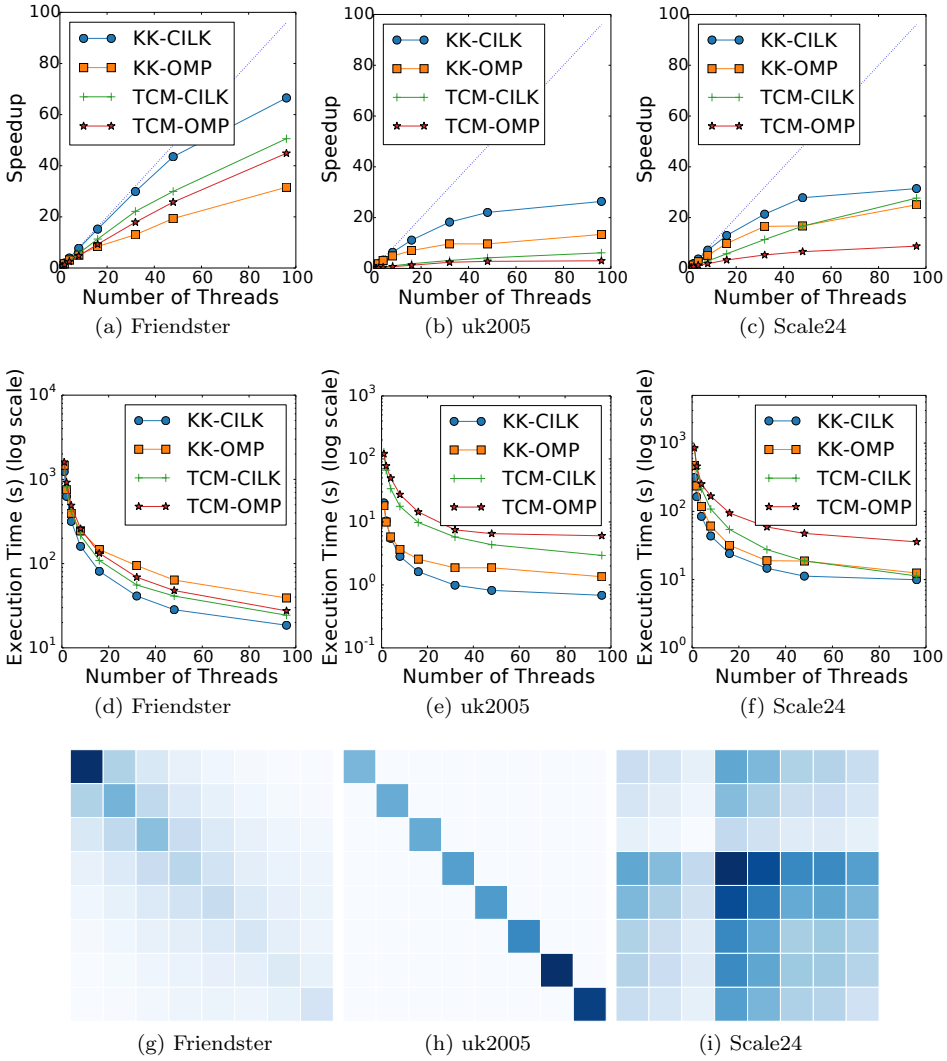
Table 4.4: Comparison between OpenMP and Cilk runtimes for three architectures. For each server KK represents KKTri/KKTri-Cilk speedup. TCM represents TCM-OpenMP/TCM-Cilk speedup. Medians of the 5 runs are used. N/A - KK-OMP/TCM implementations did not finish.

Data Set	Haswell		Skylake		KNL	
	KK	TCM	KK	TCM	KK	TCM
cit-HepTh	1.82	1.46	0.52	1.62	0.32	0.82
email-EuAll	3.38	3.05	1.86	1.86	1.04	0.69
soc-Epinions1	1.65	1.18	0.75	1.04	0.70	0.94
cit-HepPh	1.70	1.49	0.44	3.17	0.37	0.89
soc-Slashdot0811	1.93	2.03	0.73	1.23	0.67	0.96
soc-Slashdot0902	1.73	1.54	0.96	1.42	0.68	0.93
flickrEdges	1.98	2.24	1.56	3.39	0.63	2.10
amazon0312	2.04	2.26	2.29	1.89	1.61	0.99
amazon0505	2.18	2.02	2.50	1.95	1.50	1.05
amazon0601	1.60	2.82	1.15	2.20	0.45	0.98
scale18	1.72	2.63	1.17	2.42	0.74	1.70
scale19	1.29	2.65	1.23	2.49	0.92	1.89
as-Skitter	2.46	3.04	2.30	2.56	2.71	2.14
scale20	1.03	3.44	1.33	3.16	1.14	2.08
cit-Patents	2.33	3.16	1.98	1.75	1.10	1.76
scale21	1.17	2.81	1.31	3.13	1.16	1.94
soc-LiveJournal1	2.00	4.06	1.81	2.76	2.25	2.59
wb-edu	5.77	2.11	5.28	1.53	12.66	1.24
scale22	1.20	3.72	1.17	2.95	1.19	2.02
scale23	1.33	2.66	1.46	2.75	1.24	2.82
scale24	1.38	3.56	1.21	3.16	1.27	2.18
scale25	1.50	3.02	1.24	2.41	1.15	2.26
uk-2005	2.85	2.40	1.97	2.05	1.30	2.60
it-2004	2.33	18.33	2.00	16.24	0.90	15.91
twitter	1.91	2.88	1.78	2.80	N/A	N/A
friendster	2.06	1.26	2.11	1.13	N/A	N/A
uk-2007	3.86	11.04	N/A	N/A	N/A	N/A
Geometric Mean	1.87	2.61	1.42	2.33	1.07	1.65

has a very random distribution of the non-zeros (Figure 4.1i), resulting in random memory access patterns. As we can see in Figure 4.1g, Friendster graph's distribution is worse than uk-2005 and better than scale24. In this graph, there are a small number of highly dense parts and the other parts seem to present a more balanced distribution. This seems to be the best case for scalability even though the best rates are achieved in uk-2005. The poor locality in scale24 seems to hinder both the rate and scalability measures.

4.4. Relative Speedup. Figure 4.2 presents relative speedup between KKTri and TCM codes (graphs ordered by the number of edges). KKTri outperforms TCM in 23 of 27 cases. In four small instances (3 amazon graphs and emailEU), TCM performs better depending on the architecture. KKTri can achieve up to 7× speedup on graphs that have a good natural ordering such as wb-edu, uk-2005, and uk-2007 (Figure 4.2). TCM also runs out of memory for uk-2007 on two architectures. As both algorithms use the same runtime system, the primary difference can be attributed to the algorithms, data structures used, and the implementations.

4.5. Analysis of Triangle Counting Scalability. Let n and m be the number of vertices and edges in an undirected graph. If the graph is a clique, the number of triangles is bounded by $\Theta(n^3)$ and $\Theta(m^{3/2})$. These are therefore trivial worst-case bounds on the complexity of triangle *enumeration*. A trivial worst-case bound on triangle *counting* is



(a) Friendster (b) uk2005 (c) Scale24

(d) Friendster (e) uk2005 (f) Scale24

(g) Friendster (h) uk2005 (i) Scale24

Fig. 4.1: Strong scaling experiments: (a-c) x-axis presents number of threads, y-axis presents speedup with compare to best sequential algorithm. (d-f) x-axis presents number of threads, y-axis presents the execution time in seconds. (g-i) presents heat map of the graphs when we partition the graph 8 by 8 grids, darker color means more non-zeroes.

$O(n^\omega)$, where ω is the exponent of matrix multiplication. Latapy gives a summary of related complexity results and shows that triangle enumeration is bounded by $\Theta(mn^{\frac{1}{\alpha}})$ if the degree distribution is governed by a power law with exponent α [15]. This is an asymptotic improvement over the worst-case, but Berry, et al. improved upon this further by showing that triangle enumeration complexity can actually be $\Theta(n)$ in realistic circumstances [2]. These circumstances exist in many of the Graph Challenge instances, as we will show.

While the results of [2] are derived from a synthetic graph generation model (the erased configuration model), we find that they help explain empirical results from the Graph

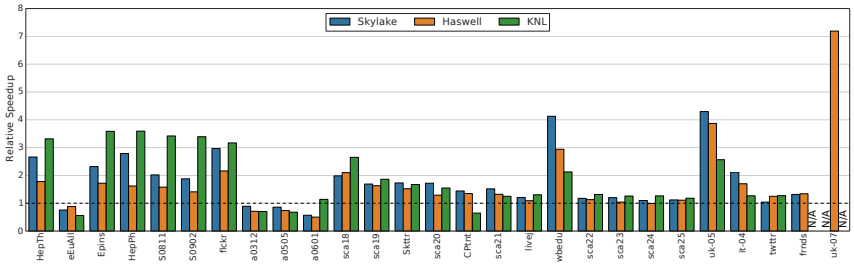
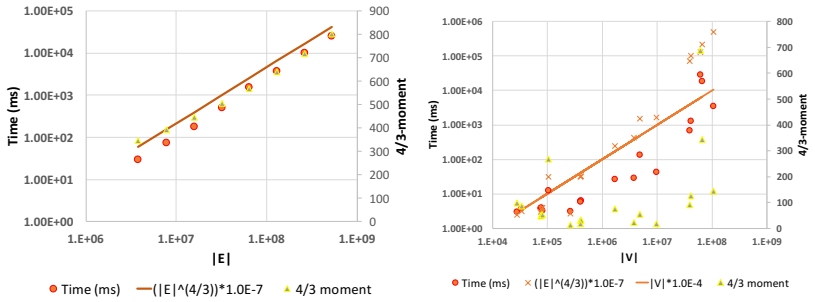


Fig. 4.2: Relative speedup in Skylake, Haswell and KNL compared to TCM. The KKTri-Cilk time for each graph in every architecture is used as the baseline. In KNL, because of memory limitations, all of the implementations failed on Friendster and uk-2007 graphs. In Skylake TCM fails on uk-2007 graph because of its memory requirement while KKTri doesn't. In the plot these cases are represented with 'N/A'.



(a) Scale Graphs (b) Non Scale Graphs

Fig. 4.3: Scale graphs and non-scale graphs triangle counting time in comparison to the edge count. Trend lines for $|E|^{(4/3)}$ and $|V|$ are also shown. The secondary axis shows the 4/3-moment. When 4/3-moments are small, we expect $O(|V|)$ performance.

Challenge graphs. The essential argument of [2] is that fast triangle enumeration is associated with small values of the 4/3-moment of the degree distribution. Letting d_v be the degree of vertex v, the 4/3-moment is defined as: $\mathbf{E}[d_v^{4/3}] = 1/n \sum_v (d_v^{4/3})$. This moment is small for many of the Graph Challenge instances.

The Graph Challenge 2017 organizers have summarized all submitted results, informally fitting their runtimes with a regression line of $O(m^{4/3})$ [18]. In this section, we use the 4/3-moment to explain the empirical runtime complexity more accurately. Graph Challenge triangle counting algorithms can be evaluated this way to ensure that they obtain optimal performance when possible.

The LU algorithm in this paper is asymptotically equivalent to the "MinBucket" algorithm analyzed in [2] and proved to be optimal when the 4/3-moment is bounded by a constant (MinBucket is alternatively called nodeIterator++ or Cohen's algorithm [6, 20, 23]). Therefore, we expect $O(n)$ performance from LU if 4/3-moments remain small as n grows.

To test this conjecture, we separate the Graph Challenge instances into two classes which we call *Scale graphs* and *Non Scale graphs*. The former comprise all of the R-MAT instances, and would include other synthetic graphs that have increasing 4/3-moments. Non-scaled

Table 4.5: Correlation coefficient between triangle counting time per vertex/edge and the $E^{\frac{4}{3}}$ or $\frac{4}{3}$-moment.

Graph Type	Time-per-vertex		Time-per-edge	
	$E^{\frac{4}{3}}$	$\frac{4}{3}$-moment	$E^{\frac{4}{3}}$	$\frac{4}{3}$-moment
Scale graphs	0.90	0.98	0.89	0.98
Non-scale graphs	0.26	0.95	0.02	0.76

graphs are all others, many of which have small 4/3-moments despite their size. We compute the 4/3-moment for all instances and calculate two correlation coefficients: that between runtime per vertex/edge and the 4/3-moment, and that between runtime per vertex/edge and $m^{4/3}$ (the function identified by the Graph Challenge organizers). Table 4.5 shows the results. Note that the per-vertex/edge runtimes are highly correlated with the 4/3-moment, as predicted by [2].

Figure 4.3 depicts the relationship between the 4/3-moment and runtime on Graph Challenge instances. Note that in the Scale graphs, $m^{4/3}$ predicts the runtime trends effectively. However, $m^{4/3}$ does not model the runtime on real graphs as effectively as the 4/3-moment does.

5. Discussion: Approximate Accumulators. To extend this work to approximately count number of triangles in a given graph we also tried to use approximate data structures; bloom filter, hyperloglog and cuckoo filter. Although, theorotically their lookup complexities are faster than our sparse hashmap accumulator their cache misses are too high for small datasets although their performance is better in larger graphs they perform really poor because of high number of cache misses. The accuracy we obtained was 99.996 %.

6. Conclusions. We developed a triangle counting method using the Cilk programming model. This implementation resulted in a rate of 10^9 on a single multicore node and was able to achieve measurable speedups compared to the OpenMP implementation of the same algorithm. This linear algebra implementation is also faster than state-of-the-art graph based implementation (up to 7×). We also showed correlation between the high rates achieved and the conductance of the graph. Finally, we were able to show that the scalability of the triangle counting is bounded by O(n) when the 4/3-moment is small. We plan to explore the use of the Cilk implementation for other algorithms such as k-truss computation.

Acknowledgments: We thank Simon Hammond, Cynthia Phillips, and Stephen Olivier for helpful discussions, and the test bed program at Sandia National Laboratories for supplying the hardware used in this paper. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

REFERENCES

[1] A. Azad, A. Buluç, and J. R. Gilbert, *Parallel triangle counting and enumeration using matrix algebra*, in Proceedings of the IPDPSW, Workshop on Graph Algorithm Building Blocks (GABB), 2015.

[2] J. W. Berry, L. K. Fostvedt, D. J. Nordman, C. A. Phillips, C. Seshadhri, and A. G. Wilson, *Why do simple algorithms for triangle enumeration work in the real world?*, in Proceedings of the 5th Conference on Innovations in Theoretical Computer Science, ITCS '14, New York, NY, USA, 2014, ACM, pp. 225–234.

[3] M. Bisson and M. Fatica, *Static graph challenge on gpu*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–8.

[4] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, *Cilk: An efficient multithreaded runtime system*, vol. 30, ACM, 1995.

[5] P. Boldi and S. Vigna, *WebGraph Datasets: Laboratory for algorithmics*. http://law.di.unimi.it/datasets.php, April 2018.

[6] J. Cohen, *Graph twiddling in a mapreduce world*, Computing in Science & Engineering, 11 (2009), pp. 29–41.

[7] T. A. Davis, *Graph algorithms via SuiteSparse:GraphBLAS: triangle counting and K-truss*, tech. rep., Texas A&M University, 2018.

[8] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software (TOMS), 38 (2011), p. 1.

[9] M. Deveci, C. Trott, and S. Rajamanickam, *Performance-portable sparse matrix-matrix multiplication for many-core architectures*, in Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International, IEEE, 2017, pp. 693–702.

[10] ———, *Multi-threaded Sparse Matrix-Matrix Multiplication for Many-Core and GPU Architectures*, arXiv preprint arXiv:1801.03065, (2018).

[11] V. Gadepally, J. Bolewski, D. Hook, D. Hutchison, B. Miller, and J. Kepner, *Graphulo: Linear Algebra Graph Kernels for NoSQL Databases*, in 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, May 2015, pp. 822–830.

[12] Y. Hu, P. Kumar, G. Swope, and H. H. Huang, *Trix: Triangle counting at extreme scale*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–7.

[13] H. Kabir and K. Madduri, *Parallel k-truss decomposition on multicore systems*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–7.

[14] KokkosKernels.

[15] M. Latapy, *Main-memory triangle computations for very large (sparse (power-law)) graphs*, Theor. Comput. Sci., 407 (2008), pp. 458–473.

[16] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*. http://snap.stanford.edu/data, 2017.

[17] R. Pearce, *Triangle counting for scale-free graphs at scale in distributed memory*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–4.

[18] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, et al., *GraphChallenge. org: Raising the Bar on Graph Analytic Performance*, arXiv preprint arXiv:1805.09675, (2018).

[19] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, D. Staheli, and J. Kepner, *Static graph challenge: Subgraph isomorphism*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, 2017.

[20] T. Schank and D. Wagner, *Finding, counting and listing all triangles in large graphs, an experimental study*, in Experimental and Efficient Algorithms, S. E. Nikoletseas, ed., Berlin, Heidelberg, 2005, Springer Berlin Heidelberg, pp. 606–609.

[21] J. Shun and K. Tangwongsan, *Multicore triangle computations without tuning*, in Data Engineering (ICDE), 2015 IEEE 31st International Conference on, IEEE, 2015, pp. 149–160.

[22] S. Smith, X. Liu, N. K. Ahmed, A. S. Tom, F. Petrini, and G. Karypis, *Truss decomposition on shared-memory parallel systems*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–6.

[23] S. Suri and S. Vassilvitskii, *Counting triangles and the curse of the last reducer*, in Proceedings of the 20th International Conference on World Wide Web, WWW '11, New York, NY, USA, 2011, ACM, pp. 607–614.

[24] A. S. Tom, N. Sundaram, N. K. Ahmed, S. Smith, S. Eyerman, M. Kodiyath, I. Hur, F. Petrini, and G. Karypis, *Exploring optimizations on shared-memory platforms for parallel triangle counting algorithms*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–7.

[25] C. E. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos, *Spectral counting of*

triangles via element-wise sparsification and triangle-based link recommendation, Social Network Analysis and Mining, 1 (2011), pp. 75–81.

[26] C. Voegele, Y.-S. Lu, S. Pai, and K. Pingali, *Parallel triangle counting and k-truss identification using graph-centric methods*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–7.

[27] N. Wang, J. Zhang, K.-L. Tan, and A. K. Tung, *On triangulation-based dense neighborhood graph discovery*, Proceedings of the VLDB Endowment, 4 (2010), pp. 58–68.

[28] M. M. Wolf, M. Deveci, J. W. Berry, S. D. Hammond, and S. Rajamanickam, *Fast linear algebra-based triangle counting with kokkoskernels*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, IEEE, 2017, pp. 1–7.

[29] A. Yaşar, S. Rajamanickam, M. Wolf, J. Berry, and Ü. V. Çatalyürek, *Fast triangle counting using cilk*, in High Performance Extreme Computing Conference (HPEC), 2017 IEEE, submitted to Graph Challenge 2018, 2018, pp. 1–7.

IMPROVING ALEGRA MEMORY USAGE

OTTO W. STRACK*, JAMES J. ELLIOTT†, CHRIS B. LUCHINI‡, AND CHRIS M. SIEFERT§

Abstract. Memory usage in the ALEGRA shock and multiphysics computational simulation code base directly impacts its users, whether it concerns the amount of physical memory used or its ability to address memory for extremely large problems (beyond 2^{32} topological entities in a finite element mesh). In this paper we discuss ways to reduce the amount of runtime memory used by ALEGRA executables, as well as their size on disk. These improvements are realized through the use of a new linker and compiler options. We address ALEGRA's ability to run extremely large problems by discussing how the code base is being transformed from 32-bit to 64-bit using a new C++ class to reference its finite elements. These improvements will allow ALEGRA to become more efficient at running simulations and run larger simulations respectively.

1. Introduction. This paper discusses two projects that improved ALEGRA memory usage. The first project involves using a new linker and compiler options to create an ALEGRA executable that is smaller and more efficient. The second project introduces a new global ordinate class in ALEGRA that enables the code base to utilize 64-bit instead of 32-bit integers for addressing the finite elements in the code. Changing the code base to utilize 64-bit is necessary because with the 32-bit code base you are limited to 2^{32} topological entities, whereas with a 64-bit base you can utilize 2^{64} topological entities.

2. Gold linker. The new linker that we added to the ALEGRA build system is called the Gold linker. The Gold linker was designed by Google [3, 4], but Firefox helped to push its development [1]. It was created by Google for a faster link time, and to provide a tool for experimenting with novel linker ideas. We use it for one of its advanced features, garbage collection. Our team's goal was to determine how much executable code we could reduce and how much memory we could save by implementing this new tool. The Gold linker also has other features, like Identical Code Folding [2]. This feature removes identical function definitions from the compiled object code. Through the use of smarter compiling and linking we were able to reduce the ALEGRA executable size and the memory used at runtime.

2.1. Making an executable. There are two major steps in creating a executable from a set of source files: compiling and linking. Compiling is the process through which the computer converts source files into object code or binary. On Linux, these object code files contain different "sections" that are used for various purposes during compilation. These sections are often labeled `.text`, `.ro`, etc... Linking can be described as merging object files into an executable format that the computer can read into its virtual and physical memory and execute (run). Even if only one source file needs to be linked, most linkers need to include some sort of library to pull in previously existing code for math or graphics. After compiling and linking an executable is produced.

2.2. Creating an executable with a smaller amount of memory. To illustrate how the Gold linker reduces memory, assume the existence of two source files, `func.cpp` and `main.cpp`. These two source files will be converted to object code using standard GNU-compiler settings, and then the object files will be linked into an executable. The first file, `func.cpp` will contain the functions `F()`, `A()`, and `C()` while `main.cpp` will have the function `main()` which uses both `F()` and `C()` from the file `func.cpp`. These are compiled into object code, also known as binary (ones and zeros) which the computer can read. A

*Eldorado High School, ottostrack@yahoo.com
†Sandia National Laboratories, jjellio@sandia.gov
‡Sandia National Laboratories, cbluchi@sandia.gov
§Sandia National Laboratories, csiefer@sandia.gov

binary copy of `func.cpp` called `func.o` will be made and likewise `main.o` will be made based on the code of `main.cpp`. If we were to look into these `.o` files we would see that they are broken into "sections". There will be numerous sections, one such being the `.text` section. This section will be the focus of our paper. When `func.cpp` is compiled all functions in the file will become binary code inside the `.text` section. The same thing will happen to the `main.cpp` file. When the linker is collecting the object code into an executable it reads the `main.cpp` file and sees that `F()` and `C()` are not in the `main.o` file, so it will define them as `weak`. In its next phase, the linker will read in the necessary files to define both `F()` and `C()` found in the function `main`. In our case, these two functions are defined in `func.cpp`. The linker will take `func.cpp` with `F()` and `C()` and insert it where `F()` and `C()` are defined as `weak` in `main.cpp`. However, because the linker inserted the entire `func.o` file, not just the needed functions, the executable now contains the function `A()`. This is a waste of space because `A()` is not necessary for a functioning executable. For clarification see figure 2.1.

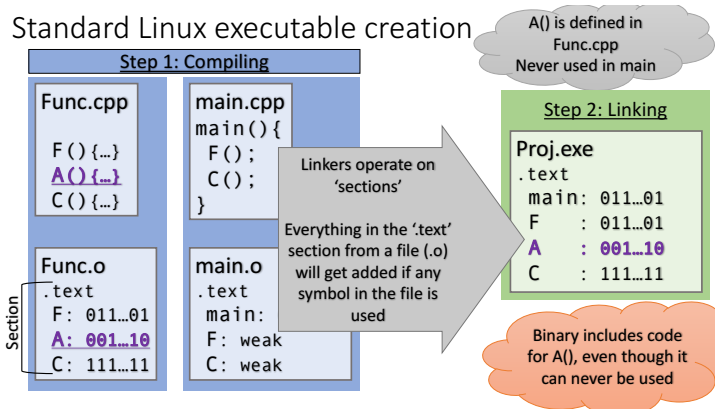


Fig. 2.1: Physical memory with A() loaded in

Now that the wastefulness of the original GNU linker is understood it will be easier to understand the process to reduce the size of the executable. There are two steps to getting a smaller executable size. We will illustrate this by using the Gold linker and adding two compiler flags. The flags `-ffunction-sections` and `-fdata-sections` force the compiler to place functions and data into separate sections. The only one that is necessary to influence the `.text` sections in the `.o` files is the `-ffunction-sections` flag. The linker was changed from the standard linker to the Gold linker. With these new optimizations we will follow the same process as described above, but now when the files are compiled the result is completely different. Because of the flags we added, instead of functions `F()`, `A()`, and `C()` all being placed into a single binary chunk under the `.text` section, now the functions all have their own `.text` section inside of `func.o`. Essentially, the compiler flags created smaller readable sections to allow the Gold linker to differentiate between parts of the object file. In contrast to before, we now see that when the linker retrieves the code needed to fill `F:weak` and `C:weak`, the linker will recognize that `.text.A` is not needed in `main()` so it will not link it

from `func.o` into `main.o`. This is where the size reduction in the executable comes from, because the wasted binary of `A()` is no longer in `proj.exe` the executable is smaller. For clarification see figure 2.2.

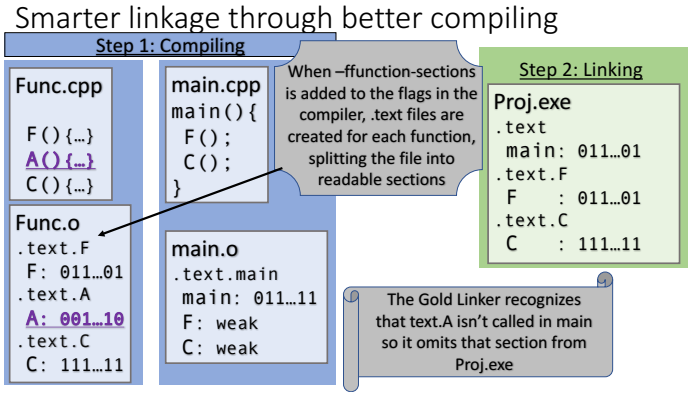


Fig. 2.2: Physical memory without A() loaded in

2.3. Smaller runtime memory usage for ALEGRA. The compiler options from the previous section result in smaller executables. Now, we discuss how this can translate into less memory used at runtime. There are two types of memory that a computer uses. The first type of memory is physical memory; This is the random access memory (RAM) of a computer or the actual existing memory. The second type of memory is called virtual memory this is the "fake" memory used to decide how to allocate physical memory. While the virtual memory loads in everything that needs to be run and allocates physical memory, the physical memory will only run small portions of the executable at a time. One function of virtual memory is to determine which parts of an executable will be loaded in with a corresponding piece of physical memory. For instance, the virtual memory may write `main()` as taking up two memory pages, so that when it is loaded into the physical memory it will load in the first page of memory with the first part of `main()` then the second part of memory with the second part of `main()`. Later, when `F()` and `C()` are called by `main()` it will load in the pages of memory that contain `F()` and `C()`. Physical memory can load in any page from the virtual memory in any order, but it can't divide the page of memory into smaller subsections.

First, consider the initial example with the standard GNU linker. When the standard GNU Executable is being loaded into virtual memory, let us assume that `F()`, `A()`, and `C()` are all loaded in next to each other in the virtual memory. `F()` and `A()` will share a page of virtual memory, but `C()` has its own page of memory. When `main()` is loaded into the physical memory it then calls for `F()` and `C()` to be loaded into physical memory as well, so all pages containing `F()` and `C()` are loaded into the physical memory. Because `F()` and `A()` were assigned the same page by the virtual memory, `A()` will also be loaded into the physical memory. Considering this, the physical memory should have four pages of memory running:

the two `main()` pages, the `F()` and the `A()` page, and the `C()` page. For clarification see figure 2.3.

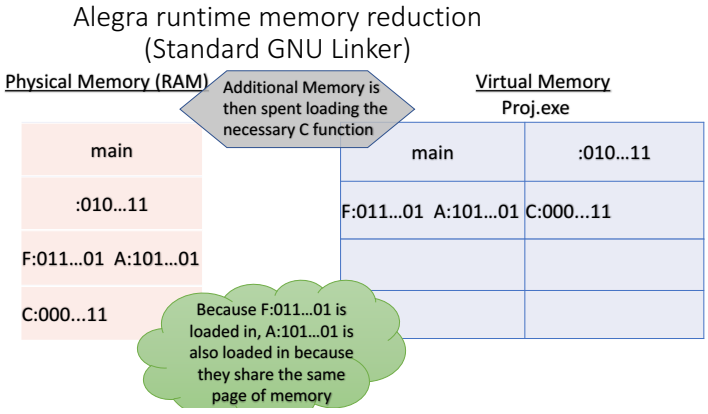


Fig. 2.3: Physical memory with A() loaded in

Next, consider the example if we use the Gold linker combined with compiler flags. When `main()` is loaded into physical memory it will call `F()` and `C()` again. However, now `F()` and `C()` share the same page of memory. This is made possible by the fact that `A()` no longer exists in the executable so it does not take up the extra space in the page with `F()`. Because `A()` is gone, an extra page of memory to get function `C()` in the physical memory is no longer needed. The computer is now only using three pages of memory instead of the four it was using with the GNU linker. This is how we achieved runtime memory savings in ALEGRA. For clarification see figure 2.4.

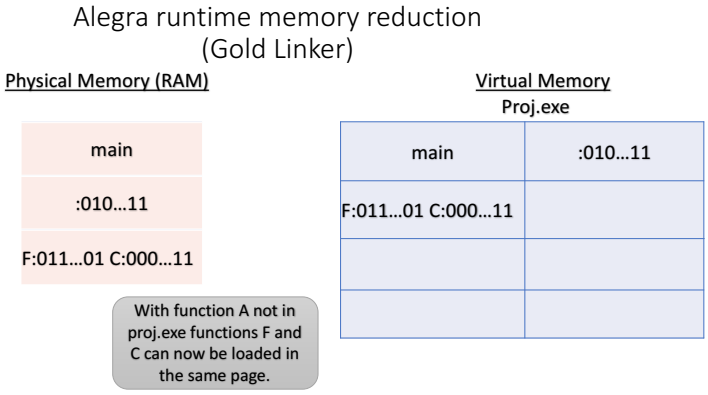


Fig. 2.4: Physical memory without A() loaded in

2.4. Gold linker results. To test the runtime memory size of ALEGRA we ran a series of benchmarks and recorded the values of `MEMORY_HEAP_PMAX`, which is the "high water" mark of memory used by the program. We found that the benchmarks used anywhere from 3-7 percent less memory than the original GNU build for both the "Comprehensive" and "Integration" ALEGRA benchmarks, as shown in figures 2.5 and 2.6, respectively.

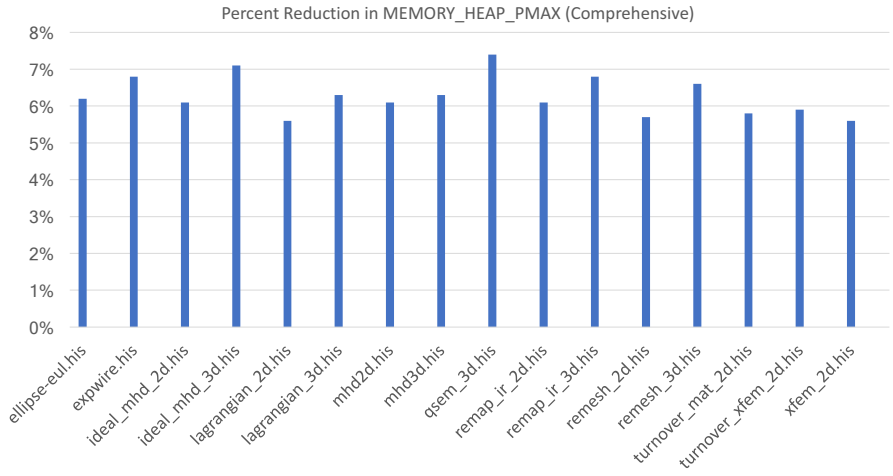


Fig. 2.5: Runtime Memory Savings (ALEGRA Comprehensive Benchmarks)

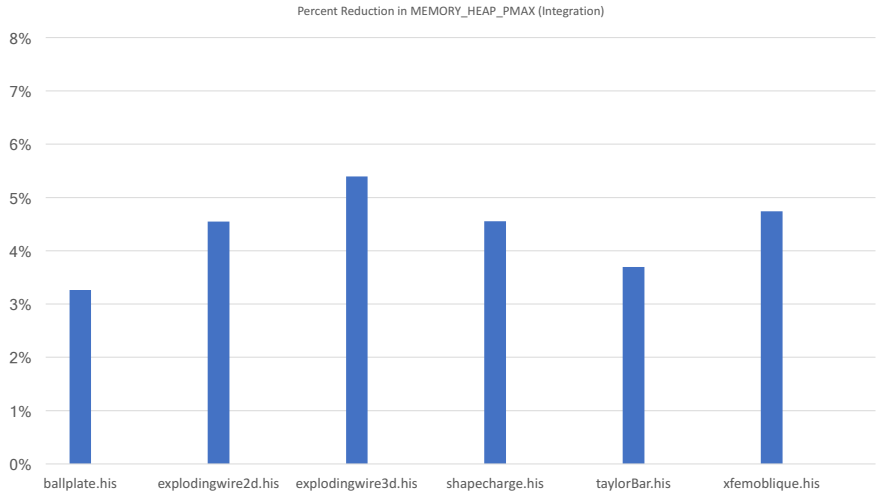


Fig. 2.6: Runtime Memory Savings (ALEGRA Integration Benchmarks)

Sheet1

	Emphasis and Alegra Builds	Size (MB)	% Smaller
2D	Alegra cts1intel Vanilla Build	183	
	Alegra cts1intel-Otto Gold Linker Build	153	16.4
	Alegra cts1intel Vanilla Debug Build	1138	
	Alegra cts1intel-Otto Gold Linker Debug Build	1075	5.5
	Emphasis cts1intel Vanilla Build	164	
	Emphasis cts1intel-Otto Gold Linker Build	136	17.1
	Emphasis cts1intel Vanilla Debug Build	953	
	Emphasis cts1intel-Otto Gold Linker Build	902	5.4
	Alegra cts1intel Stripped Vanilla Build	136	
	Alegra cts1intel-Otto Stripped Gold Linker Build	110	19.1
	Emphasis cts1intel Stripped Vanilla Build	119	
	Emphasis cts1intel-Otto Stripped Gold Linker Build	95	20.2
3D	Alegra cts1intel Vanilla Build	264	
	Alegra cts1intel-Otto Gold Linker Build	217	17.8
	Alegra cts1intel Vanilla Debug Build	1731	
	Alegra cts1intel-Otto Gold Linker Debug Build	1636	5.5
	Emphasis cts1intel Vanilla Build	174	
	Emphasis cts1intel-Otto Gold Linker Build	144	17.2
	Emphasis cts1intel Vanilla Debug Build	1057	
	Emphasis cts1intel-Otto Gold Linker Build	1001	5.3
	Alegra cts1intel Stripped Vanilla Build	199	
	Alegra cts1intel-Otto Stripped Gold Linker Build	158	20.6
	Emphasis cts1intel Stripped Vanilla Build	128	
	Emphasis cts1intel-Otto Stripped Gold Linker Build	102	20.3

Fig. 2.7: Executable Memory Savings (Various ALEGRA Builds)

The ALEGRA executables we created with the Gold linker are much smaller than those created with the standard GNU linker. Between the standard builds of ALEGRA, whether they are the 2D, 3D, Emphasis, or stripped versions of them, we saw anywhere from a 15-20 percent reduction of size in megabytes (MB). The "stripped" build is the binary with all the debug and line information removed. See figure 2.7. The debug versions and their corresponding Gold versions also had a reduction in executable size. The Gold versions were on average about 5-6 percent smaller than the debug builds (also seen in figure 2.7).

2.5. Gold linker overview. The Gold linker experiment was a chance for the ALEGRA team to achieve savings in both executable size and runtime memory size at no cost to other facets of the code. We saw a substantial reduction in the size of executables. Saving approximately 15-20 percent for standard builds of ALEGRA. Additionally, we observed a 3-7 percent decrease in the runtime memory usage.

3. Global Ordinal Project. The Global Ordinal Project was conceived so ALEGRA could run simulations with over 2^{32} elements in a single problem on a finite mesh. Previously when simulations were attempted that exceeded 2^{64} elements the program would give incorrect results or crash. The way to increase the number of elements that the executable can reference has to do with the type of integers used to address memory. Before the Global Ordinal Project was started, the ALEGRA code was based on the standard `int` type. Our team changed this to type `long long`, which makes code 64-bit instead of 32-bit. Although the most elegant way to achieve this is to create a `typedef long long`, our team wanted to ensure that we had not missed any 32-bit integer types that needed to be converted, so we made a `typedef` named `ANGO_t` and inserted it in a class called `ANGO` (as seen below). Because `long long ANGO` is in a class, whenever the compiler tries to implicitly convert an `ANGO` object to an `int` the compiler gives an error that states "type `ANGO` could not be converted to type `int`". To start the process of converting the code base we changed a few `global int`s to `ANGO`s and allow the errors to appear. We then proceeded to follow the errors fixing them then building the code again, slowly making our way through the code base converting `int`s to `ANGO`s. This method ensured that we did not accidentally miss any `int`s that needed converting. Although the class is being kept for developmental purposes, once we are confident that our code is correct we will delete it and replace it with a `typedef`. A partial listing of the `ANGO` class is provided below.

```
class ANGO {
 public:
  typedef long long ANGO_t;
  typedef unsigned long long uANGO_t;
  typedef long  ANGO_fake_t;
  typedef int CLASSIC_GO_t;
 private:
  ANGO_t go_;
 public:
  ANGO() { go_ = 0l;}

  ANGO (ANGO_t  g) { assert(g <= INT_MAX && g > INT_MIN); go_ = g;}

  ANGO operator-() const {
    ANGO n;
    n.go_ = -go_;
    return n;
```

```
  }

  ANGO_t getGOValue() const

  friend std::ostream& operator<<(std::ostream& os, const ANGO& );
};

inline std::ostream & operator<<(std::ostream& os, const ANGO & g) {
  assert(g.go_ < INT_MAX && g.go_ > INT_MIN);
  os<<g.go_;
  return os;
}
```

4. Conclusion. We discussed improvements in memory usage of the ALEGRA code base using a better set of compiler flags and the Gold linker. There were two memory improvements concerning the implementation of the Gold linker. The first was a smaller executable size and the second was a smaller amount of physical memory used (RAM). We also discussed the Global Ordinal Project, which will eventually allow users to run simulations with over two billion elements.

REFERENCES

[1] *Optimizing real-world applications with GCC Link Time Optimization*, 2010. http://gcc.gnu.org/wiki/summit2010?action=AttachFile&do=get&target=hubicka-slides.pdf.

[2] *Safe ICF: Pointer Safe and Unwinding Aware Identical Code Folding in Gold*, 2010. http://gcc.gnu.org/wiki/summit2010?action=AttachFile&do=view&target=tallam.pdf.

[3] S. Tallam, C. Coutant, I. Taylor, X. Li, and C. Demetriou, *Gold*, (2008). https://www.airs.com/ian/gold-slides.pdf.

[4] I. Taylor, *Google releases new and improved gcc linker*, (2008). https://opensource.googleblog.com/2008/04/gold-google-releases-new-and-improved.html.

EFFICIENT POINT MERGE USING DATA PARALLEL TECHNIQUES

ABHISHEK D. YENPURE* AND KENNETH D. MORELAND†

Abstract. We study the problem of merging three-dimensional points in space. We present a fast, efficient approach that uses data parallel techniques for execution in various shared memory environments. We then compare our approach against methods of a widely used scientific visualization library accompanied with a performance study that shows our approach works well with different kinds of parallel hardware (many-core CPUs and NVIDIA GPUs), and data sets of various sizes. We also present a heuristic for efficiently clustering spatially close points together, which is one reason our method performs well against the other methods.

1. Introduction. With this work, we contribute a many-core algorithm for merging nearby points for scientific visualization. The primary challenge for this algorithm is efficiently identifying which points are close to each other. Although the point merge algorithm does not receive as much attention as algorithms like iso-surfacing or volume rendering, it is used regularly in scientific visualization tools.

Point merging typically complements visualization algorithms that iterate over cells like iso-surfacing or slicing. These algorithms iterate over input cells to generate triangles; the vertices of these output triangles are interpolated and not part of the source data set. Consider a case where an iso-surface operation is applied on two neighboring cells, C_1 and C_2, to produce two abutting triangles T_1 and T_2, as shown in Figure 1.1. T_1 comprises vertices V_{11}, V_{12}, and V_{13}, and T_2 is composed of vertices V_{21}, V_{22}, and V_{23}. As abutting triangles, two sets of their vertices should be coincident. Without loss of generality, assume V_{11} and V_{21} are equal and V_{12} and V_{22} are equal. If these pairs are not merged, then connectivity-based operations will fail. In particular, rendering this triangle data will result in flat shading, since the normal of V_{11} would reflect only T_1 (and not T_2), the normal of V_{21} would reflect only T_2 (and not T_1), and V_{12} and V_{22} would suffer similarly. However, if V_{11} and V_{21} are

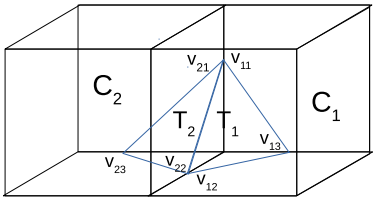


Fig. 1.1: Example of abutting triangles generated by applying an iso-surface operation.

merged to make a new point V' and if V_{12} and V_{22} are merged to make a new point V'', then the lighting will appear smooth.

Point merging is useful in other settings as well. When visualization algorithms generate triangles with small areas, then lighting issues can again arise (among other issues). The vertices of triangles with small areas are typically very close to one another and thus can be reduced to a single vertex through point merging. Further, numerical errors can sometimes cause interpolated vertex positions to be slightly offset. So in the previous iso-surface example, even vertices that should be considered exactly coincident might have inexact point coordinates. Point merging solves this problem as well.

*University of Oregon, abhishek@uoregon.edu
†Sandia National Laboratories, kmorel@sandia.gov

Point merging is typically done by organizing points into a spatial data structure and traversing that data structure to locate nearby points. Our algorithms works in this vein, although we are able to arrange our operations so that no explicit data structure is needed. It is for this reason that we refer to our approach as "Virtual Grid" point merging. A particular focus for our algorithm is on many-core architectures. Our code is designed to use the parallel building blocks available from the VTK-m library that ensure good performance over varying architectures. We evaluate this code, comparing to another module in VTK-m, and to both parallel and serial modules in VTK. Overall, we find that our Virtual Grid point merging algorithm is competitive with other parallel point merging techniques, more resilient to irregular distributions of input points, scalable within shared memory domain, and performs well on both CPU and GPU devices.

2. Related Work. We divide this section into two parts In the first part we discuss previous works which deal with merging of points. In the second part we discuss previous works for developing scientific visualization algorithms using data-parallel techniques.

2.1. Merging Points. This section describes previous works for merging spatially close points. Many of these works present approaches for merging points in a related application of mesh simplification, in which case points are referred to as vertices. This section is further divided into three parts. In the first part we discuss works that use search structures to discover spatially points. In the second part we discuss works that use much simpler data structures to merge points. In the third part we discuss works that focus on improving the accuracy of merging points.

2.1.1. Merging Points Using Search Structures. Rock and Wozny [19] provided an approach for reconstructing the topology of a model. The first step in their approach involved merging of spatially close points, which they term 'vertex merging,' as the points that they merge are vertices of triangular facets. To locate close points that need to be merged, they used an AVL search tree constructed with the vertices of the facets. Kanaya et al. [6] presented a similar approach where they perform vertex clustering using an octree, and offer multiple degrees of mesh simplification. Further, they use a depth first approach on this octree to locate the connected components at the desired degree of simplification, which are simplified then by vertex merging.

2.1.2. Merging Points Using Spatial Binning. The construction of search structures for very large data is known to have a significant computational overhead. Rossignac and Borrel [20] address this issue with a mesh simplification approach that uniformly subdivides a 3D volume containing the input mesh into smaller 3D regions, or "bins." All vertices that fall within the same region are merged together. The input mesh is then corrected to remove all degeneracies that were introduced by the merge operation, yielding a simplified output mesh. While this approach is fast, it has very little control over the accuracy and quality of the simplification. Shin et al. [21] modify the uniform binning approach of Rossignac and Borrel to observe an increased accuracy of vertex merging. To increase the merging accuracy, vertices were merged only if they existed within a user-defined ϵ-tolerance of each other. Vertices within this tolerance may be stored in one of the adjacent cells associated with the bin in which the vertex resides. To limit searching in all adjacent cells, Shin et al. describe a way to decompose a cell in multiple regions. This modification significantly reduces the number of cells that must be sequentially searched. The primary focus of this work is on the construction of topology from triangle data, given no prior connectivity information.

2.1.3. Improving Accuracy of Point Merging. Another set of works focus on minimizing the error introduced by merging vertices. Garland and Heckbert [5] perform

vertex merging by contracting pairs of vertices, one pair at a time. A pair of vertices are merged based on a per-vertex error cost function that calculates the sum of square distances to the planes of the triangles that meet at the vertex. To merge a pair (v_1, v_2), the position v_{new} is calculated as the minimum-error point. Low and Tan [12] achieve a more-consistent mesh simplification via a cell clustering approach that assign weights to vertices based on the probability of the vertex lying on the mesh boundary from an arbitrary viewing direction, and on the size of the faces bounded by the vertex. Lindstrom and Turk [11] minimize the impact of point merging on the geometric properties (e.g., area and volume) of the mesh by assigning a cost weights to the mesh edges, where the cost is a function of the volume, boundary, and shape preservation properties of the mesh vertices. Finally, Barequet and Kumar [1] merge a pair of vertices exist on different edges. In this method, edges are selected based on the cost of moving the endpoint vertex of one edge to the endpoint vertex of another edge. This cost has a user-provided upper-bound ϵ, and a pair of vertices are merged by averaging their coordinates.

2.2. Data-Parallel Techniques. Recently, a growing body of literature has investigated the design of data-parallel algorithms for scientific visualization applications using data-parallel primitive (DPP) [2] operations, such as sort, gather, scatter, map, reduce, copy, etc. These approaches inspire techniques for performing point merging in a data-parallel setting. Lessley et al. [8] describe approaches for data-parallel searching for duplicate elements in a 3D mesh topology. Their work finds the external faces of a given data set by hashing the three vertices that define a triangular face. These hashed values are then used to group duplicate faces. Point merging is a similar problem, whereby coincident points hash to the same value. The approaches used to handle hash collisions directly affect the correctness and the performance of point merging. Although the work of Lessley et al. does not readily provide a solution for merging points, its results provide motivation to use data-parallel techniques.

Miller et al. [13] present a data-parallel method that generates the output topology of a visualization operator using the knowledge of the input mesh topology. One of the visualization operators studied was the marching cubes algorithm, whereby all output vertices occur on the edges of the cells of the input voxel grid. This work used the unique identifies of these edges in the input mesh as keys to store the newly-calculated points as values. These key-value pairs are used to merge points of the output topology using a reduce-by-key DPP. However, this approach cannot be directly applied to a mesh when the information about the original topology is unavailable.

Finally, the emergence of platform-portable libraries, such as NVIDIA Thrust [17] and VTK-m [16], have made it convenient to write algorithms in a single code implementation for execution across multiple platforms (e.g., both CPUs and GPUs). These high-level libraries provide a set of core DPP that are optimized for each different target platform of execution using low-level, platform-specific libraries, such as OpenMP, Intel TBB, and NVIDIA Thrust. An algorithm is then written in terms of these core DPP or user-defined DPP ("worklets"). Since its release a couple of years ago, platform-portable algorithms written with VTK-m have demonstrated competitive performance to algorithms designed for a specific platform [7, 9, 10, 18]. This collection of work justifies the merit of the platform-portable framework. In this work, we design our algorithm with VTK-m and contribute the implementation as a standard, open-source filter within the library.

3. Formal Definition of Point Merging. Informally, the point merging operation finds all point within a distance δ and merges those points together. However, it is necessary to define correctness of merging points in ambiguous cases, when we have tolerance $\delta \neq 0$. Figure 3.1 presents a scenario, where we have 3 points to be merged together, points A, B,

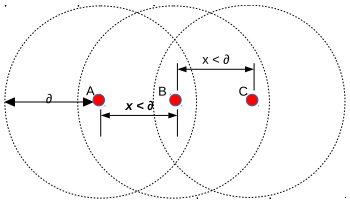
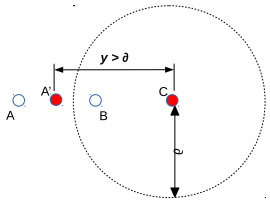
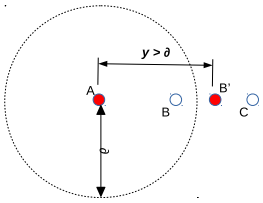


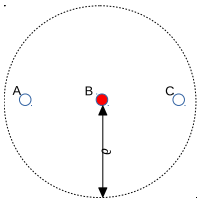
Fig. 3.1: Points to be merged, where $x < \delta$



(a) Merge points A, and B. $y > \delta$



(b) Merge points B, and C. $y > \delta$



(c) Merge points A, B, and C

Fig. 3.2: Possible, correct solutions to merging points

and C. Points A and B satisfy the distance criteria, and points B and C satisfy the distance criteria with respect to tolerance δ, but A and C do not satisfy the distance criteria. This is an ambiguous case because it is unclear whether A, and C should be merged together to satisfy all of B's distance criteria. In this work we consider any group of points that satisfy the following properties to be a proper point merge.

1. In a group of points to be merged, every point in the group is within distance δ from at least one point in the group.
2. A group of points to be merged have a fully connected distance criteria relationship among the points. That is between any two points there is a path of points that satisfy the distance criteria between them.
3. After the merging is complete, none of the resulting points are within distance δ from one another.

Note that our definition of point merging allows for multiple, equally correct solutions. For example, Figures 3.2a, 3.2b, 3.2c present all the possible, correct solutions to merge the points from Figure 3.1. Since there are multiple correct solutions to merge points successfully in an ambiguous case, the outputs of different approaches can be slightly different from each other, i.e they might not contain the same points in the final output. However, they must satisfy the discussed properties.

4. Technical Approach. This section provides details on our Virtual Grid approach. The key differentiator between our approach and previous approaches is that our approach is designed for many-core architectures and is composed of data parallel techniques. At a high level, our approach works as follows. It begins by binning points into the cells of a 3D uniform grid, which is the same first step taken by previous approaches. However, previous approaches have relied on an explicit mapping from cells to lists of points within the cells. Our approach, instead, focuses on using data parallel operations to rearrange a large list of points. After rearranging, each thread of a many-core device operates on one

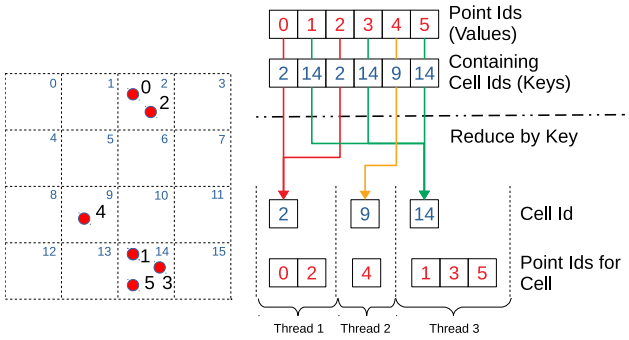


Fig. 4.1: Notional example of working of the Virtual Grid approach.

cell at a time, and the points that lie within that cell are readily available in the thread's memory. This is the reason we term our approach as a "Virtual Grid" approach. Figure 4.1 provides a 2D schematic of how this process occurs in our Virtual Grid approach. Another difference between our algorithm and previous approaches is in the merge step. Many previous approaches have merged all points that occur in the same cell, and also looked at neighboring cells for close points in adjacent cells. Our approach works differently; we perform distance computations between pairs of points within a cell to ensure that only the points that occur within the tolerance δ are merged. However, this operation by itself is not sufficient to merge all points.

Occasionally, close points (i.e., points that are with δ and should be merged) are mapped into adjacent cells of the virtual grid. It is important to identify when this happens and ensure they are merged. We deal with this case by performing the multiple iterations of our point merge operation, with each iteration having the virtual grid offset by a small amount. This way, the points that satisfy the tolerance criteria, but were binned into different cells in one iteration, can be binned into the same cell in a later iteration. Specifically, we run eight iterations. We calculate the eight bounding boxes of the eight virtual grids as:

1. Expanding the bounding box of the original data set by δ in all directions (1 *count*);
2. Shifting the bounding box from step 1 by $\delta/2$ along each axis (3 *counts*);
3. Shifting the bounding box from step 1 by $\delta/2$ diagonally along planes XY, YZ, and XZ (3 *counts*); and
4. Shifting the bounding box from step 1 by $\delta/2$ along the diagonal of the bounding box (1 *count*).

Our algorithm also treats a tolerance of 0 as a special case, and only a single iteration is required to merge coincident points, since the issue of close points in adjacent cells is not relevant in this case.

A single iteration of our approach works as follows:

1. For each point, identify which cell in the Virtual Grid contains it.
2. Rearrange the layout of the points so that points contained in the same cells are grouped together in an array. This is done via a Reduce-By-Key data parallel operation, which is provided by VTK-m.
3. For each group, calculate the points that are within the tolerance δ of each other. For a set of points that satisfy the tolerance criteria, output neighbor index as the point with the minimum index in the set.
4. Cluster all points that share the same neighbor, which becomes a neighborhood.

This is done via a Reduce-By-Key data parallel operation, which is provided by VTK-m.

5. Reduce each cluster of points by calculating its centroid. The set of reduced clusters is the output of the merge operation.

Miller et al. [13] show how to perform Steps 2 and 4 efficiently in parallel by sorting, and VTK-m provides this grouping as a basic feature [14]. The steps described above are performed once for all the virtual grids described earlier. Algorithm 3 explains how each of these steps is performed. This approach performs at its best when the cells of the virtual

Algorithm 3 Virtual Grid approach

1: **for** all $i \leftarrow 0, : Points :$ in parallel **do** ▷ Step 1
2: $Cell[i] \leftarrow$ GET-CELL-FOR-POINT$(()Points[i])$
3: **end for**
4: $Bins \leftarrow \{(bin, P_{bin}) : P_{bin}$ are indices of all points in $bin\}$ ▷ Step 2 : Performed by VTK-m, using $Cell$ array from Step 1.
5: **for** all $(bin, P_{bin}) \in Bins$, do in parallel **do** ▷ Step 3
6: **for do**$i \in P_{bin}$
7: $nearest[i] \leftarrow i$
8: $dist \leftarrow \infty$
9: **for do**$j \in P_{bin}$ and $j \neq i$
10: $j\_dist \leftarrow$ DISTANCE$(Points[i], Points[j])$
11: **if then**$j\_dist \leq \delta$ and $j\_dist \leq dist$ and $j < i$
12: $nearest[i] \leftarrow j$
13: $dist \leftarrow j\_dist$
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: $Clusters \leftarrow \{(cluster, P_{cls}) : P_{cls}$ are indices of all points in $cluster\}$ ▷ Step 4: Performed by VTK-m, using $Nearest$ array from Step 3.
19: **for** all $(cluster, P_{cls}) \in Clusters$, do in parallel **do** ▷ Step 5
20: $centroid \leftarrow$ GET-CENTROID$(Points, P_{cls})$
21: **end for**

grid have small numbers of points binned into them. Binning fewer points in the cells of the virtual grid translates to performing fewer computations to correctly group points that satisfy the tolerance δ. Also, since the virtual grid is represented sparsely in memory, i.e., no state information for the grid is stored, it can use much smaller bins and therefore reduce the cost of searching for neighbors in step 3 of Algorithm 3. To achieve this, we calculate the optimal dimensions for the uniform grid based on the tolerance δ using the following equation:

$$dimension_t = \frac{length_t}{2 \times \delta} \tag{4.1}$$

where $dimension_t$ is the dimension of the grid along axis t and $length_t$ is the length of the bounding box along axis t in the original data set. This enables us to create the smallest bins for which all pairs of points within the tolerance δ of each other are guaranteed to bin in the same bin or in adjacent bins. However, if the tolerance δ is very small, then the dimensions of the virtual grid can become so large that it requires more than 2^{32}, or 2^{64}

bins, which means they cannot be indexed by 32-bit or 64-bit numbers. In such cases, we limit the dimensions of the grid using the following equation:

$$dimension_t = floor(\sqrt[3]{|MaxValue|}) \tag{4.2}$$

Where $MaxValue$ is the maximum number of bins that can be indexed using the data type we choose for binning points.

Section 6 presents impacts of the data type choice on performance.

5. Experimental Overview. To better compare and study the performance characteristics of our methods, we performed tests with the following variables:

- 6 algorithms;
- 3 data sets;
- 2 values of tolerance δ; and
- 2 different hardware architectures.

We ran performance tests with selective combinations of these variables Table 5.1 details these combinations. Note that we tested each of these combinations with all three of our datasets. In total we compared 168 unique combinations of these variables. Limitations of the features from the VTK library restricted us from testing all 288 possible combinations. We elicit these limitations further in this section. Our performance evaluations are presented in Section 6.

	Tolerance		Parallel			
Algorithm	$\delta = 0$	$\delta \neq 0$	Serial	TBB(6)	CUDA	Count
vtkMergePoints	✓		✓			3
vtkCleanPolyData		✓	✓			3
vtkSMPMergePoints	✓			✓		18
VTK-m Point Locator	✓	✓	✓	✓	✓	48
VTK-m Virtual Grid (32 bit)	✓	✓	✓	✓	✓	48
VTK-m Virtual Grid (64 bit)	✓	✓	✓	✓	✓	48
Total						168

Table 5.1: Combinations that we tested. To interpret this table, consider the row for "VTK-m Point Locator," which supports execution with both the values of tolerance, and all types of parallelism. For different hardware architectures we ran 1 experiment for each, serial and GPU execution, and performed 6 experiments for TBB with varying number of CPU threads, making the total 8. For this case, we have 2 *tolerances* $\times$ 8 *parallelizations* $\times$ 3 *data sets* $=$ 48 *experiments*.

5.1. Algorithms. We consider a total of six algorithms: three algorithms are from the VTK library, and three algorithms are implemented using VTK-m.

5.1.1. VTK Algorithms. The three algorithms from VTK come from the vtkMergePoints, vtkSMPMergePoints, and vtkCleanPolyData classes. Each of these classes has limitations. The "vtkMergePoints" point locator merges exactly coincident points, but does not work in parallel. We also considered "vtkSMPMergePoints," which performs the same operation in parallel[1], but requires the user to explicitly take care of threading. Finally,

[1]We encountered a bug in this module, and so the experiments performed in our study came from a revised version provided by Kitware, The bug fix will be available in a future version of VTK. The bug report

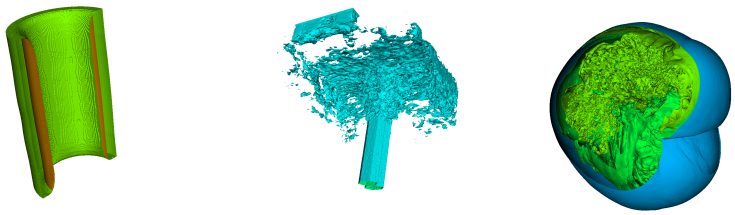
we studied the "vtkCleanPolyData" filter, which is capable of merging points within a user provided tolerance. This filter also performs additional tasks, such as removing degenerate triangles. Like vtkMergePoints, vtkCleanPolyData filter operates only in serial.

5.1.2. VTK-m Algorithms. From the three algorithms implemented in VTK-m, two of the algorithms come from variants of our Virtual Grid approach, and one other algorithm comes from a point locator based approach for merging points.

The variants in the Virtual Grid approach result from the user's choice in data type to store the bin indices while binning points. For this paper, we chose to test with two different data types: 32-bit integers and 64-bit integers.

The point locator based approach was implemented as a VTK-m counterpart for the point locator based modules in VTK, namely vtkMergePoints and vtkSMPMergePoints. The solution for this study was implemented using a modification of the "PointLocatorUniformGrid" module from the VTK-m library, which uses spatial binning to locate the nearest neighbor of a points. The modification was performed in order to support querying nearest neighbors of the same points that were used to build the search structure. Without this modification, for a given query point, the points locator returns the same query point as the nearest neighbor. Other point locator modules based on a k-D tree and on two-level uniform grid are also available in the VTK-m library but were not used for this study. The algorithm that was developed uses multiple iterations to merge points. For each iteration it finds the nearest neighbor for every point, and merges the neighbors by reducing them to their centroid. These iterations are performed until no new neighbors are discovered in the the set of the residual points.

5.2. Data Sets.



(a) Fusion (b) Thermal Hydraulics (c) Supernova

Fig. 5.1: Data sets used for the performance tests. An additional clip operation was applied to generate the images for the Fusion and Supernova data to reveal more intricate details.

We explored three data sets for this study. In each case, we took an existing data set and applied an iso-surface operation. This fusion data set comes from the NIMROD [22] simulation code which is used to model the behavior of burning plasma. This thermal hydraulics data set comes from the NEK5000 [4] code which is used for the simulation computational fluid dynamics. Finally, the supernova data set comes from a supernova simulation made available by Blondin and Mezzacappa [3]. We used the VTK-m library to perform the iso-surfacing and disabled the option for merging coincident points. This resulted in a triangle soup, where no triangles shared any common vertices. Explicitly, if there were N triangles, then the soup would have $3 \times N$ vertices, with many of the vertices

can be found at the link: https://gitlab.kitware.com/vtk/vtk/issues/17386

being coincident and replicated in the vertex list. Figure 5.1 provides the visuals for the test data sets, and Table 5.2 provides additional information for reproducibility purposes.

		Output		
Data set	Iso-Values	Points	Cells	Bounds
Fusion	2.4, 3.2	4125540	1375180	0 – 1
Thermal Hydraulics	42, 64	15686430	5228810	0 – 1
Supernova	0.02, 0.05, 0.07	24493224	8164408	0 – 431

Table 5.2: Details of the data sets we used for our experiments.

5.3. Tolerance for Merge. We test our algorithms with two distinct values for tolerance, $\delta = 0$, and $\delta = 0.0001$. These two values provide a fair comparison with features provided by the VTK library and also enable us to study the performance characteristics of our algorithms.

$\delta =$ ***0 :*** We use a zero tolerance to merge exactly coincident points present in the data set. Choosing the tolerance as $\delta = 0$ enables us to compare our parallel algorithms implemented using VTK-m with the parallel features to merge coincident points from the VTK library.

$\delta =$ ***0.0001 :*** We use a non-zero tolerance δ to merge points that occur within the specified distance of each other. This enables us to test cases where there is a need to merge points that may not be exactly coincident, but are separated by some threshold distance between them. For our tests we used the tolerance $\delta = 0.0001$. As mentioned earlier, the VTK library has no alternatives for merging points that are separated by some difference in parallel. We instead compare our VTK-m implementations with the 'vtkCleanPolyData' filter provided by the VTK library, which also accepts a tolerance value from the user to merge points in the data set, but only supports serial execution.

5.4. Hardware Architectures. The biggest advantage of using the VTK-m library is its ability to provide portable performance over multiple architectures that are comparable to platform specific solutions. We tested our implementations on two different hardware environments:

CPU: Dual Intel Xeon Platinum 8160 CPU of the SkyLake family, each with 24 cores running at 2.10 GHz, able to run 2 threads per core, and equipped with 191 GBytes of DDR4 memory.

GPU: NVIDIA Tesla P100 GPU of the Pascal family with 3584 CUDA cores, capable of 5.3 TeraFLOPS of double precision computations, and equipped with 16 GBytes of HBM2 memory.

VTK-m uses the TBB (Thread Building Blocks) library as a backend threading library for execution on many core CPUs, and uses CUDA as a backend threading library for execution on NVIDIA GPUs. Henceforth, all parallel CPU execution times are reported using TBB, and GPU execution times are reported using CUDA.

6. Results. We divide this study into four parts. In Section 6.1 we present the comparison of our Virtual Grid approach with the other algorithms from Section 5.1 for merging exactly coincident points ($\delta = 0$). In Section 6.2 we present the comparison of our Virtual Grid approach with the other algorithms from Section 5.1 for merging close

points within a tolerance ($\delta = 0.0001$). In Section 6.3 we compare the strong scaling characteristics of our Virtual Grid approach while executing on multi-core CPUs. In Section 6.4 we present the performance portability of the our Virtual Grid approach through VTK-m while executing on multi-core CPUs and NVIDIA GPUs. In Section 6.3 and Section 6.4, we quantify performance and scaling in terms of the rate of processing points, which is calculated using the following equation:

$$processing\ rate = \frac{number\ of\ points}{execution\ time\ (seconds)} \tag{6.1}$$

This gives us the total number of points that a program is able to process in a second. We use processing rate as a measure because it is a better indicator of parallel speed-up than execution time [15].

				VTK-m	
	Parallel		Point	Virtual Grid	
Data Set	Environ.	VTK	Locator	32-bit	64-bit
Fusion	Serial	0.54	4.48	1.43	1.35
	TBB	0.12	0.26	0.15	0.20
	CUDA		0.08	0.04	0.05
Thermal	Serial	9.61	68.22	6.37	6.03
Hydraulics	TBB	0.64	2.13	0.58	0.73
	CUDA		0.82	0.16	0.18
Supernova	Serial	7.64	55.82	10.24	9.71
	TBB	0.68	2.16	0.93	1.05
	CUDA		0.78	0.25	0.28

Table 6.1: Execution times in seconds for merging points with $\delta = 0$. The execution times for TBB are presented using 48 CPU cores.

6.1. Merging Coincident Points ($\delta = 0$). In this part of the study we compare the execution times of all algorithms for merging bit-wise coincident points. The serial execution times for the VTK library come from the "vtkMergePoints" module. The TBB execution times for the VTK library come from a custom program that explicitly manages spawning of threads for the "vtkSMPMergePoints" module. As the VTK library does not support execution on GPUs, there are no CUDA results for VTK.

The two selected VTK modules for this part of the study are imperfect comparators. Both VTK modules only deal with the special case of bit-wise coincident points. The algorithms that we implemented using the VTK-m library deal with the general case, which means they need to perform more computations. Specifically, we perform additional distance computations to make sure we group points only within the given tolerance δ. Also, we calculate the centroid for all the points grouped together to merge them, coincident or otherwise. Despite this asymmetry, we feel the comparison is valuable since it lets us test against another parallel implementation.

Table 6.1 presents the execution times for this part of the study. In most cases the execution times for the serial and TBB versions of the VTK library are better than our Virtual Grid approach. Again, this is an expected result as we perform significantly more computations in our Virtual Grid approach. The point locator based approach, implemented

in VTK-m, performs slower than the other approaches in all cases. The reason for this behavior, apart from the extra computations, is the cost associated with the construction of the point locator data structure for every iteration until the merge converges.

Table 6.1 also shows the difference between the two versions of our Virtual Grid approach. Under most circumstances, the version using 32-bit integers performs better than the one using 64-bit integers. The 32-bit integers results in larger bins, which results in more points binned together, which in turn results in more time spent computing distances between points. However, 32-bit integers can be sorted faster than 64-bit integers, and the reduced sort time more than compensates for the extra distance comparisons, which are embarrassingly parallel.

When considering performance across data sets, the algorithms behaved differently. The execution times for the Virtual Grid are consistently proportional to the number of points being processed. For example, the Virtual Grid algorithm using 32-bit integers running with TBB processes at a consistent rate of about 27 million points per second for all three data sets. In contrast, the algorithms based on point locators, which includes the VTK algorithm, process points at a much slower rate for thermal hydraulics data than the other two data sets. This is because the thermal hydraulics data is not spatially distributed as evenly as the other two data sets. We observe, therefore, that the Virtual Gird approach is more resilient to the spatial distribution of the points in the data and a better choice for unstructured data at different scales.

			VTK-m		
	Parallel		Point	Virtual Grid	
Data Set	Environ.	VTK	Locator	32-bit	64-bit
Fusion	Serial	2.17	5.21	1.97	2.0402
	TBB		0.33	0.29	0.3605
	CUDA		0.10	0.07	0.1148
Thermal	Serial	64.94	77.23	10.52	11.03
Hydraulics	TBB		2.26	1.22	1.41
	CUDA		1.03	0.33	0.37
Supernova	Serial	54.02	57.20	13.65	15.94
	TBB		2.38	1.25	1.95
	CUDA		0.84	0.35	0.48

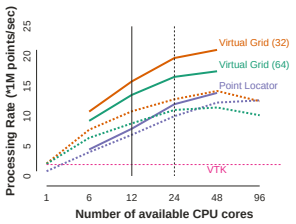
Table 6.2: Execution times in seconds for merging points with $\delta = 0.0001$. The execution times for TBB are presented using 48 CPU cores.

6.2. Merging Close Points ($\delta = 0.0001$). In this part of the study we compare the execution times of all algorithms for merging points that are separated by a small tolerance. The execution times for the VTK library come from the "vtkCleanPolyData" module. This is the only module available in the VTK library that readily supports merging of points that are separated by some distance. As mentioned in Section 5, it does not support parallel execution on any architecture. In addition to merging close points, the vtkCleanPolyData module removes degenerate triangles; we modified our algorithms implemented using VTK-m to also remove these triangles to have a fair comparison.

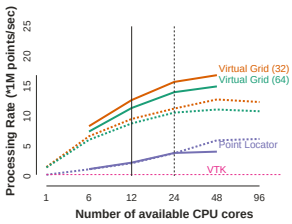
Table 6.2 presents the execution times for this part of the study. There is a significant increase in the execution times for both the VTK library and the VTK-m implementations

compared to the times presented in Section 6.1. The increase in execution time for the VTK library are a result of now having to perform distance checks and calculating the centroid for merging points that are within the distance δ. The modules for merging coincident points from the VTK library only perform a comparison for equality. The increase in the execution times for the VTK-m implementations are a result of having to perform multiple iterations to merge points that are within the distance δ. The Virtual Grid approach in VTK-m is the most performant of all the algorithms that we tested for this case. As we optimally bin points into small cells of the virtual grid, we only have to perform a few distance checks between pairs of points that belong to the same cell in order to merge them correctly. This reduces the computations that are performed by each thread when executing in parallel. Also, in contrast to the point locator based algorithm implemented in VTK-m, the Virtual Grid approach does not have the overhead of maintaining and updating a search structure to locate close points.

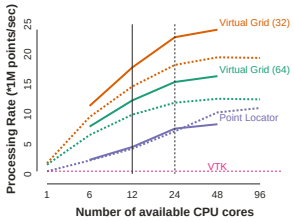
6.3. CPU Scaling Study.



(a) CPU strong scaling for fusion data.



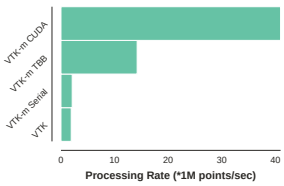
(b) CPU strong scaling for thermal hydraulics data.



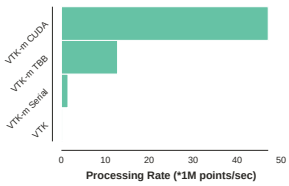
(c) CPU strong scaling for supernova data.

Fig. 6.1: CPU strong scaling for merging close points ($\delta = 0.0001$) : The dashed lines show scaling results when using both NUMA nodes, and the solid lines show scaling results when threads are restricted to a single NUMA region. Beyond the dashed vertical line at 48 threads we use logical CPU cores of both the NUMA nodes, and beyond the solid vertical line at 24 cores we use logical CPU cores for the chosen NUMA region. The VTK data shown is the processing rate for the serial version.

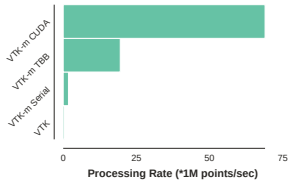
In addition to comparing our Virtual Grid approach to the reference VTK counterparts, and the point locator based VTK-m algorithm, we also studied the strong scaling characteristic of the algorithms for merging close points. Further, since the CPU hardware has two processors using a NUMA architecture, we also study the impacts of the NUMA architecture on the performance of our implementations. Figure 6.1 plots the results for this part of the study.



(a) Processing rates for fusion data



(b) Processing rates for thermal hydraulics data



(c) Processing rates for supernova data

Fig. 6.2: Comparison of processing rates for Virtual Grid (32-bit) approach across devices. These rates are based on the tolerance $\delta = 0.0001$. The TBB results are provided using 48 CPU cores.

Since there is no parallel module in VTK for this task, the plots show a flat line for the corresponding VTK module. The plots present the comparison of strong scaling results of our implementations when CPU threads use both NUMA regions and when threads are restricted to use a single NUMA region. In the presented plots, the dotted lines show the strong scaling results when the programs use both NUMA nodes and the solid lines show the strong scaling results when the programs execute on a single NUMA node using only the local memory of that node. The scaling results for a single NUMA node only go to 48 cores, because a single node only contains 24 physical cores, which could also be used as 48 logical cores.

In general, the plots indicate that our VTK-m implementations are able to benefit from an increase in the number of available CPU cores. That said, we observe that while this scaling is good, it does level off as we approach the number of cores available. We believe this is because the point merging algorithm, which by its nature must load points from disparate memory locations, is memory bandwidth bound. Consequently, we observe diminishing returns for more cores, almost no benefit from engaging hyper-threading, and a large penalty when executing across NUMA memory regions. These observations suggest:

1. Increasing the number of threads for execution beyond the available physical CPU cores does not result in any performance improvements for our implementations;
2. Limiting threads to use a single NUMA node and its local memory provides the best rate for processing points.

These ideas can be used as a heuristic to schedule programs to yield the best performance. If users were to use our implementations in a distributed memory setting, they should launch processes such that each process is limited to execute in a single NUMA region, with threads not exceeding the maximum available physical cores.

6.4. Performance Across Devices. Figure 6.2 presents the comparison of processing rates for the Virtual Grid approach across devices. The processing rate for the VTK library comes from the "vtkCleanPolyGrid" filter and is used as a comparator for the Virtual Grid approach when executing in serial. This section reinforces our findings from Section 6.2 that the Virtual Gird approach is very efficient for a general case of merging points. When executing in serial the Virtual Grid approach offers better throughput compared to the VTK comparator. Additionally, the Virtual Grid approach is able to use available parallelism efficiently. When executing on GPUs the Virtual Grid approach is able to achieve a much higher processing rate.

7. Conclusions. We presented our Virtual Grid algorithm, a fast and scalable solution for merging points in a data set that occur within a certain user provided tolerance. This algorithm leverages VTK-m and its data parallel techniques.

Overall, our algorithm performs well on varied parallel environments and workloads. Our comparison against parallel VTK showed that the Virtual Grid approach was competitive. Although our implementation is slower in some cases, we attribute this to the fact that the VTK code is designed and optimized for a single specific use case and thus removes several operations necessary for our more general code. We also compared against another VTK-m module, point locators, and found that we had superior performance in all cases.

For future work, we expect further optimizations are possible with respect to optimizing memory accesses. We will also be exploring alternatives to the sort used for grouping points in the same Virtual Grid bins. We are considering hash-table–based approaches similar to that used by Lessley et al. for external facelist calculation [7, 9].

REFERENCES

[1] G. Barequet and S. Kumar, *Repairing cad models*, in Proceedings. Visualization '97 (Cat. No. 97CB36155), Oct 1997, pp. 363–370.
[2] G. E. Blelloch, *Vector models for data-parallel computing*, vol. 75, MIT press Cambridge, 1990.
[3] J. M. Blondin, A. Mezzacappa, and C. DeMarino, *Stability of standing accretion shocks, with an eye toward core-collapse supernovae*, The Astrophysical Journal, 584 (2003), pp. 971–980.
[4] P. Fischer, J. Lottes, D. Pointer, and A. Siegel, *Petascale algorithms for reactor hydrodynamics*, in Journal of Physics: Conference Series, vol. 125, IOP Publishing, 2008, p. 012076.
[5] M. Garland and P. S. Heckbert, *Surface simplification using quadric error metrics*, in Proceedings of the 24th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216.
[6] T. Kanaya, Y. Teshima, K.-i. Kobori, and K. Nishio, *A topology-preserving polygonal simplification using vertex clustering*, in Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia, ACM, 2005, pp. 117–120.
[7] B. Lessley, R. Binyahib, R. Maynard, and H. Childs, *External facelist calculation with data-parallel primitives*, in Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization, Eurographics Association, 2016, pp. 11–20.
[8] B. Lessley, K. Moreland, M. Larsen, and H. Childs, *Techniques for data-parallel searching for duplicate elements*, in Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on, IEEE, 2017, pp. 1–5.
[9] B. Lessley, T. Perciano, M. Mathai, H. Childs, and E. W. Bethel, *Maximal clique enumeration with data-parallel primitives*, in Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on, IEEE, 2017, pp. 16–25.
[10] S. Li, M. Larsen, J. Clyne, and H. Childs, *Performance impacts of in situ wavelet compression on scientific simulations*, in Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization, ISAV'17, New York, NY, USA, 2017, ACM, pp. 37–41.
[11] P. Lindstrom and G. Turk, *Fast and memory efficient polygonal simplification*, in Visualization'98. Proceedings, IEEE, 1998, pp. 279–286.
[12] K.-L. Low and T.-S. Tan, *Model simplification using vertex-clustering*, in Proceedings of the 1997 symposium on Interactive 3D graphics, ACM, 1997, pp. 75–ff.

[13] R. Miller, K. Moreland, and K.-L. Ma, *Finely-threaded history-based topology computation*, in Proceedings of the 14th Eurographics Symposium on Parallel Graphics and Visualization, Eurographics Association, 2014, pp. 41–48.

[14] K. Moreland, *VTK-m user's guide (version 1.1)*, Tech. Rep. SAND 2018-0475 B, Sandia National Laboratories, 2018.

[15] K. Moreland and R. Oldfield, *Formal metrics for large-scale parallel performance*, in International Conference on High Performance Computing, Springer, 2015, pp. 488–496.

[16] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. Ma, H. Childs, M. Larsen, C. Chen, R. Maynard, and B. Geveci, *Vtk-m: Accelerating the visualization toolkit for massively threaded architectures*, IEEE Computer Graphics and Applications, 36 (2016), pp. 48–58.

[17] NVIDIA Corporation, *Thrust*, Nov. 2018. https://developer.nvidia.com/thrust.

[18] D. Pugmire, A. Yenpure, M. Kim, J. Kress, R. Maynard, H. Childs, and B. Hentschel, *Performance-portable particle advection with vtk-m*, in Eurographics Symposium on Parallel Graphics and Visualization, H. Childs and F. Cucchietti, eds., The Eurographics Association, 2018.

[19] S. J. Rock and M. J. Wozny, *Generating topological information from a bucket of facets*, in 1992 International Solid Freeform Fabrication Symposium, 1992.

[20] J. Rossignac and P. Borrel, *Multi-resolution 3d approximations for rendering complex scenes*, in Modeling in computer graphics, Springer, 1993, pp. 455–465.

[21] H. Shin, J. C. Park, B. K. Choi, Y. C. Chung, and S. Rhee, *Efficient topology construction from triangle soup*, in Geometric Modeling and Processing, 2004. Proceedings, IEEE, 2004, pp. 359–364.

[22] C. Sovinec, A. Glasser, T. Gianakon, D. Barnes, R. Nebel, S. Kruger, D. Schnack, S. Plimpton, A. Tarditi, M. Chu, et al., *Nonlinear magnetohydrodynamics simulation using high-order finite elements*, Journal of Computational Physics, 195 (2004), pp. 355–386.

PREDICTING DISK FAILURE WITH MACHINE LEARNING

BRANDON L. MORRIS[†] AND MATTHEW L. CURRY[‡]

Abstract. Hard drives are one of the least reliable components of a large-scale system and are prone to failure. However, it can be difficult to determine when a disk is close to failure and for what reason. Almost all disks report various S.M.A.R.T. statuses that provide data about various health-related attributes of the disk. Even with a disk's S.M.A.R.T. data, understanding how likely a disk is to fail is nontrivial. We tackle the problem of disk failure prediction using machine learning. Specifically, we train a random forest classification model on raw S.M.A.R.T. attributes from a dataset of over 120,000 disks across 88 models and six manufacturers. Despite the large number of different disk types and the massive data imbalance, our resulting model achieves good results with a test precision of 0.97 and F1 score of 0.75.

1. Introduction. Disk failure is a common obstacle in large data centers. Reacting to failure reduces system availability and can even cause data loss, which can have serious impacts on the center's operation. Furthermore, disk failure is reasonably common inside large centers with thousands of disks. To make matters worse, disks often fail with little or no warning. Detecting when disks are likely to soon fail is a difficult challenge and a solution can provide substantial benefit to maintaining large data centers.

Nearly all disks contain Self-Monitoring, Analysis, and Reporting Technology (S.M.A.R.T.) statuses that report various statuses about the current drive. For instance, S.M.A.R.T. 9 reports the number of power-on hours. In addition to raw values, the statuses also have normalized variants that are meant to be more interpretable. However, some S.M.A.R.T. attributes are ill-defined and can vary between manufacturers.

Unfortunately, it can be difficult to reliably and accurately anticipate a disk's failure directly from its S.M.A.R.T. statuses [9]. A common method involves defining a threshold for certain attributes, and disks that report values exceeding that threshold are replaced. Which attributes are tracked and the values for the thresholds are arbitrary, leading to a system that almost certainly has a high false positive rate and in turn unnecessarily replaces disks.

In this report, we describe our efforts to apply machine learning methods to predict the failure of a disk directly from the reported S.M.A.R.T. attributes. Specifically, we use a random forest classifier trained directly on raw S.M.A.R.T. values from a very large dataset of over 120,000 disks from various manufacturers. Our classifier makes very accurate predictions, with a test-time precision of 0.97 and an F1 score of 0.75.

2. Related Work. S.M.A.R.T. statuses have often been used as an indication of the health and potential failure of a disk. As mentioned earlier, it is common for large data centers to implement simple heuristic-based schemes such as thresholding specific attributes [1]. More relevant to this work, the problem of anticipating disk failure can be approached as a machine learning problem.

Some of the earliest attempts worked with datasets much smaller than commonly found in modern machine learning and that used in this work. These approaches included variations of a naive Bayes classifier [4, 9] and statistical tests [5]. The goal of classifying failure has also been rephrased as an anomaly detection problem [4], which confronts the substantial imbalance hard drive failure data.

More recent approaches utilize decision tree-based learning methods, which is also done in this work. These algorithms include regularized greedy forests [2] and boosted random forests [11]. Similar to our work, Li et al. [7] used tree-based models to classify a disk's

[†]Arizona State University, brandonmorris@asu.edu
[‡]Sandia National Laboratories, mlcurry@sandia.gov

failure, and even goes further by attempting to regress a numeric "health value" for a given disk.

Other machine learning algorithms have been applied to disk failure prediction. Yang et al. [12] demonstrated effective results with logistic regression and a very large dataset. Deep recurrent neural networks have also been applied to the regression problem of attempting to predict the remaining life of a disk [3, 11].

We note that in all of the previous works mentioned, the datasets used for training and evaluating the model only include disks of a single model, or a handful of models from the same manufacturer. However, we trained our model on the full range of disks and models available, totaling 88 unique models from six manufacturers. To the best of our knowledge, we are the first to train jointly across such a multitude of models and manufacturers and achieve useful results.

3. Using Random Forests to Predict Disk Failure. Our goal was to create a machine learning model that predicts impending failure in disks from S.M.A.R.T. attributes. Here we describe our strategy for collecting and processing the data, as well as the architectural decisions for the classifier itself.

3.1. The Backblaze Dataset. To train our model we leveraged the existing and public Backblaze dataset [1]. Backblaze takes daily snapshots of a subset of S.M.A.R.T. attributes for every disk in their production data center. In addition, they include contextual information such as the disk model and its capacity. The complete dataset dates back to 2013, but for our purposes we only leveraged data from 2015 up through the first quarter of 2018. We chose not to include the first two years for both computational reasons and out of concern that their data distributions may not be similar to the latter years from our initial testing. The resulting dataset utilized for training and evaluation contains 126,386 different disks with numerous models and manufacturers.

The data is arranged in columnar format, where each row represents a daily snapshot of a single disk, and the columns correspond to S.M.A.R.T. attributes or relevant disk information. Both normalized and raw values are recorded in the Backblaze dataset. Only a subset of all S.M.A.R.T. attributes are recorded, with some added over time. We only examined attributes present in the entire dataset, for a total of 39 attributes. Backblaze also records a single bit for whether a disk has failed, which is particularly relevant to our study.

3.2. Data Cleaning and Preparation. The data comes from Backblaze as a series of CSV files. Given the structured nature of the format, it was particularly well suited for manipulation using the Pandas[1] library [8].

Manually inspecting the data guided our process for cleaning and preparing. A small number of rows contained no useful information. This was likely due to an error during data collection and those values were removed from the dataset. In addition, we noticed that Backblaze would commonly remove disks that had not actually failed. A disk could have been retired because (i) it was replaced by a larger, faster disk, or (ii) the disk's S.M.A.R.T. attributes signaled a potential failure in Backblaze's own monitoring system [6]. Since it is impossible to determine the cause of retirement from the data, we removed all rows corresponding to retired disks to avoid biasing our model.

Our initial analysis also revealed a heavy imbalance in the data. The vast majority of disks do not fail during the time the data was collected. This is likely compounded by Backblaze's retirement strategy. Leaving this imbalance unaddressed can seriously hamper the performance of a machine learning model. We investigated several sampling methods

[1]The Pandas library requires that all the data fit into memory. While this was not an issue during out testing, Dask is an alternative that alleviates this requirement with a very similar API.

during training to attempt to balance the dataset, but ultimately found that a preprocessing technique we call "failure smoothing" had the best effect. Failure smoothing involves going back n days prior to a disk's failure and marking those rows as failures as well. This helps leverage information in the S.M.A.R.T. statuses leading up to a failure. Failure smoothing also makes sense in practice, since we often want to predict a failure *prior* to a disk actually ceasing to function, rather than right when it happens. Failure smoothing is similar to window averaging strategies used in other approaches to this problem [2]. In our tests we set n to be 10.

In addition to the above preparation, we also performed typical cleaning common to nearly all data science problems. Categorical features like the disk model and capacity were mapped to integers. Missing raw S.M.A.R.T. values were replaced with the median value for that attribute, since raw values are typically discrete integer counts. Similarly, missing normalized values were replaced with the mean value for that attribute.

3.3. The Classification Model. We experimented with several classification models, and found best results with a random forest classifier. A random forest is a collection of individual decision trees, trained on subsets of the data. Their predictions are averaged leading to better generalization than an individual decision tree. We used a standard implementation from Scikit-Learn library [10].

Given the imbalance of our dataset, the selection of a performance metric was carefully considered. Normal accuracy is not particularly informative in our case, since a naïve model that always predicts a non-failure would achieve very high accuracy. Instead we evaluated our models using the F1 metric, which is the harmonic mean between precision and recall. Precision measures the accuracy with which the model makes positive predictions, and is defined by the following equation:

$$\text{precision} = \frac{tp}{tp + fp} \tag{3.1}$$

where tp and fp correspond to the number of true positive and false positive predictions respectively. A model's recall measures how effective the model is at recognizing positive instances, and is defined as

$$\text{recall} = \frac{tp}{tp + fn} \tag{3.2}$$

where fn is the number of false negative predictions. Combining these, we can define the F1 metric:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.3}$$

Like precision and recall, the F1 score ranges between 0 and 1, with higher value indicating a better performing model.

4. Results. We were able to successfully train a random forest classifier on the raw S.M.A.R.T. statistics and achieve good results. Our best performing model achieved an F1 score of 0.75 on a hold-out test set of data not utilized during training. Looking closer, we saw the resulting model had a very high precision (0.97) and a less impressive recall (0.61). This suggests that, when our model makes a failure prediction, it is almost always correct, but it struggles to catch every failure. This is reasonable, since disks can fail with little to no indication in their S.M.A.R.T. values. Additionally, the high precision value is particularly useful in real applications, since a low false positive rate means failure predictions can be taken with more confidence.

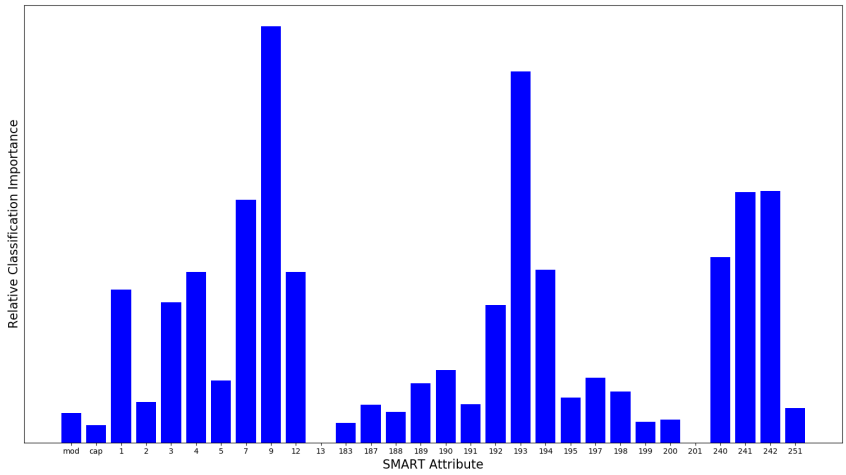


Fig. 4.1: The relative importance of S.M.A.R.T. attributes for failure classification. A higher bar indicated a more significant contribution to the final classification status for a particular input. Note that S.M.A.R.T. attributes 13 and 201 have no significance; this is likely because most of the values for those attributes were missing in the dataset.

While developing our final model, we experimented with different data subsets and hyperparameter values. For instance, we found the increasing the number of decision trees in the random forest improved the results, but with diminishing returns. We used 150 trees in our testing, as it gave good results with a reasonable training time. Additionally, the dataset contains two values for each S.M.A.R.T. attribute: a raw and normalized value. We trained our random forest on both values and each one individually. The results are depicted in Table 4.1. Using both values slightly underperformed compared to just the raw values, and training with just the normalized values gave very poor test results. This could be because the range for raw values is substantially wider, or because the manufacturers in our dataset do not normalize S.M.A.R.T. values identically.

During our experimentation, we found that the selection of S.M.A.R.T. attributes to train with had the single greatest impact on validation accuracy. Backblaze lists five attributes that they have determined useful for predicting failure[2], but we found that these were not adequate to produce a useful model. Training on just these attributes only gave an F1 score of 0.09. We tried a slightly larger subset, and the results improved. We only got our best results by training on nearly every attribute provided in the dataset. We also examined the relative significance of each attribute when making the classification prediction. These importances are graphed in Fig. 4.1. It seems that most attributes are at least somewhat significant to the model, the most import attributes being the power on hours (9), the load cycle count (193), and the total LBA read/written (241 and 242).

[2] https://www.backblaze.com/blog/what-smart-stats-indicate-hard-drive-failures/

Metric	Raw	Normalized	Both
Precision	0.972	0.202	0.966
Recall	0.606	0.21	0.591
F1	0.745	0.206	0.733

Table 4.1: Comparison of training with different presentations of the S.M.A.R.T. statistics. We achieved best results only using the raw S.M.A.R.T. values. Only using normalized values gave poor results, possibly because of the mix of manufacturers in the training set. The random forest was able give good results with value types, but still less than the raw, and at the cost of extra memory consumption.

5. Future Work. There are several avenues for continuing the work described in this report. An obvious route would be to regress the length of the remaining life for a disk based on its S.M.A.R.T. statuses. This may need to be done in conjunction with a classifier such as the model described here, since the vast majority of disks will have very long lives. Additionally, our model could be combined with other techniques in an ensemble. Unsupervised techniques such as clustering could be incorporated to better identify potential failures and improve overall recall when used in conjunction with our high-precision random forest.

We also believe our model has interesting implications for real-world system maintenance and design. Being able to simulate system conditions and infer the likelihood of failure is very valuable to understanding the reliability of the overall system. In addition, our model can easily be incorporated into a real-time monitoring system that can identify potentially problematic disks and automatically replicate the disk's contents and notify operators.

6. Conclusion. In this work, we leveraged the power of machine learning to develop a simple and efficient model to predict the failure of a disk directly from its raw S.M.A.R.T. values. The resulting random forest classifier is performant with a very low false positive rate, despite the mix of numerous disk models and manufacturers. Our best model achieved a precision of 0.97 and a final F1 score of 0.75 on a hold-out test set. In addition, our results can be very practically used in real situations where disk failure needs to be anticipated. We hope that these results and our account of attaining them inspire others to take advantage of machine learning in their own domains.

REFERENCES

[1] *Hard drive data and stats.* https://www.backblaze.com/b2/hard-drive-test-data.html.
[2] M. M. Botezatu, I. Giurgiu, J. Bogojeska, and D. Wiesmann, *Predicting disk replacement towards reliable data centers*, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 39–48.
[3] F. D. dos Santos Lima, G. M. R. Amaral, L. G. de Moura Leite, J. P. P. Gomes, and J. de Castro Machado, *Predicting failures in hard drives with lstm networks*, in 2017 Brazilian Conference on Intelligent Systems (BRACIS), IEEE, 2017, pp. 222–227.
[4] G. Hamerly, C. Elkan, et al., *Bayesian approaches to failure prediction for disk drives*, in ICML, vol. 1, 2001, pp. 202–209.
[5] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, *Improved disk-drive failure warnings*, IEEE transactions on reliability, 51 (2002), pp. 350–357.
[6] A. Klein, *What smart stats tell us about hard drives.* https://www.backblaze.com/blog/what-smart-stats-indicate-hard-drive-failures/, 2016.
[7] J. Li, X. Ji, Y. Jia, B. Zhu, G. Wang, Z. Li, and X. Liu, *Hard drive failure prediction using classification and regression trees*, in Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on, IEEE, 2014, pp. 383–394.

[8] W. McKinney et al., *Data structures for statistical computing in python*, in Proceedings of the 9th Python in Science Conference, vol. 445, Austin, TX, 2010, pp. 51–56.

[9] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, *Machine learning methods for predicting failures in hard drives: A multiple-instance application*, Journal of Machine Learning Research, 6 (2005), pp. 783–816.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[11] G. Shu and J. E. Jang, *Proactive storage health management to reduce data center downtime (general machine learning)*, (2017).

[12] W. Yang, D. Hu, Y. Liu, S. Wang, and T. Jiang, *Hard drive failure prediction using big data*, in Reliable Distributed Systems Workshop (SRDSW), 2015 IEEE 34th Symposium on, IEEE, 2015, pp. 13–18.

WHAT'S THE STITCH? AN EFFICIENT I/O LIBRARY FOR EXTENDING LENGTH SCALES IN MULTISCALE MODELING

ENZE CHEN[‡], JAY LOFSTEAD[§], AND JOHN A. MITCHELL[¶]

Abstract. One of the foremost challenges in multiscale modeling of manufacturing materials is bridging the gap between the length scales of material transformations (microns or less) and that of engineered parts (centimeters or more). While more advanced models are constantly being developed to meet this challenge, novel data storage and I/O methods have received less attention, even though I/O can be a severe bottleneck for parallel applications that read and write large volumes of data. In this report, the *stitch* library is introduced. *Stitch* is an efficient I/O API and database format that enables out-of-core computations and merges outputs written over time to construct the full simulation domain; in essence, *stitch* builds the simulation domain analogously to the way an additive manufactured (AM) part is built. The new *stitch* capability is demonstrated by applying it to AM simulations using *spparks*, which is an open source kinetic Monte Carlo simulation framework particularly suited for grain growth simulations associated with welding and AM. Scaling studies are conducted, profiled, and compared to existing *spparks* I/O capabilities. It is shown that *stitch* not only has comparable scaling performance to other I/O methods, but also enables increased data storage flexibility and larger simulation domains with significantly fewer computational resources.

1. Introduction. In exploring the central tenet of process-structure-property-performance relationships in materials science, researchers are more frequently turning towards computational models and numerical simulation techniques. While there has been an explosion of progress in recent years developing computational models for material systems [6], one of the principal existing challenges is resolving length scale disparities between material transformations at the sub-micron level and engineering performance at the scale of centimeters and above. Formulating such multiscale models is an active area of research [1,20], with the ultimate goal being high-fidelity simulations that can scale across space and time. In this report, we focus on this length scale challenge for additive manufacturing (AM) and work towards resolving it using the *stitch* database and library API.

While the focus has been on improved models and algorithms, a major component of performant large-scale simulations is efficient data storage and I/O. Existing I/O libraries such as ADIOS [9], NetCDF [15], and HDF-5 [18] all assume the entire simulation domain is written as output by the application at each time step. While this may be necessary for certain applications, there are other scenarios where it would be more efficient to only compute on and handle data in a locally affected region while leaving the rest of the domain, which doesn't change, untouched. Therefore, the development of an I/O library that can flexibly manipulate local data and later merge the results to reconstruct the full simulation domain is a critical step towards enabling practical multiscale models for these styles of applications.

To highlight the urgency of this problem, we focus on the rapidly emerging field of metal AM [3]. AM is the process of joining materials in a layer-by-layer fashion from a wire feed or powder bed using a laser beam, as opposed to subtractive manufacturing methodologies that machine away material and/or manipulate bulk structures to form a final part geometry. Because material macroscopic properties (moduli, yield strength, ductility, etc.) of finished parts are highly dependent on underlying microstructures, modeling of the AM process is a crucial step for design and qualification of AM built parts. During the AM process, evolution of metallic grains primarily occurs in the vicinity of the laser beam heat source, while the system outside of this zone is largely unaffected; the *stitch* concept was born from this basic

[‡]Stanford University, enze@stanford.edu
[§]Sandia National Laboratories, gflofst@sandia.gov
[¶]Sandia National Laboratories, jamitch@sandia.gov

fact. *Stitch* facilitates simulation of the AM build by constructing the computational domain analogously to the way it is constructed in practice. As novel AM alloys [10] and increasingly larger designs [7] emerge, simulation technologies for AM must be capable of predicting microstructure on the entire part; in this report, *stitch* is introduced as an approach to handling this problem.

While many AM models for metals exist in the literature [2], we focus on the kinetic Monte Carlo (KMC) code *spparks* [13,14]. KMC is a popular simulation technique that spans the gap between atomistic and continuum length scales [19]; *spparks* has been previously used to simulate grain growth [5] and various welding and AM applications [16,17]. For the welding application, *spparks* moves a teardrop-shaped melt pool for a specified number of Monte Carlo steps (MCS) and evolves grains within a certain distance of pool boundary. The affected region is termed the *heat affected zone* (*haz*); grains outside this region are assumed to be unaffected by the moving heat source. While *spparks* has existing built-in AM models and I/O capabilities, practical AM capabilities are limited by the aforementioned problems. Existing *spparks* models require the entire domain to exist in memory despite the fact that grain growth is highly localized; furthermore, I/O is also conducted on the entire domain but output files (text or binary) are not very easy to work with for post-processing.

In this report, we highlight the initial development and application of the *stitch* I/O library, an efficient I/O tool optimized for applications such as those found in *spparks*. The *stitch* library is linked with *spparks* to enable more flexible I/O and facilitates data analysis and image creation after AM process simulations are completed. *Stitch* enables out-of-core computation by only recording data in the area that is changing, thus reducing memory, storage footprint, and required computational resources for large-scale simulations. Computations are on a localized simulation domain which is rectangular and referred to as the computational volume (*cv*); the *cv* contains the melt pool, *haz*, and local weld path. Using *stitch* for parallel calculations is simplified because a single *stitch* file can be written by multiple processes; after a series of *spparks* calculations are completed, one *stitch* file contains all results, thus simplifying post-processing and image generation. Data in the *stitch* file can be accessed through a C API or Python API for further calculations and analysis. To assess the performance of this I/O method, we perform strong and weak scaling studies for a typical-sized simulation domain and compare the performance to that of existing capabilities in Section 3.

One of the primary innovations afforded by the *stitch* I/O library, a process we call *stitching*, is best illuminated by the AM weld application. Traditionally, a *spparks* user would specify dimensions of the full simulation domain, which is loaded into memory for simulation; subsequently the entire domain is written as output using the traditional approach. Incorporating the full domain into memory is both computationally expensive and wasteful, since the majority of *spparks* sites at each time step lie outside the *haz* and are unchanged. Not only does *stitch* enable simulation over just the local *cv*, but it can also merge outputs written over time to assemble a picture of the entire simulation domain at any given time. We call this process *stitching* and illustrate it in Figure 1.1. At time $t = 0$, an external script calculates the dimensions of cv_1 and initializes the microstructure. The melt pool welds across cv_1 where only sites within cv_1 are considered for computation and I/O. When the pool reaches the end of cv_1 ($t = 20$), a second overlapping cv_2 is initialized for the pool to continue moving, where now only data in cv_2 is used for computation and I/O. This *stitching* process continues until the full simulation domain is covered or stitched together. When the user later queries the full simulation domain (or any subset thereof) at any time step, *stitch* is capable of merging previously written blocks in a way that forms a coherent picture of the specified extent. As an example, the welded microstructure is shown across the full domain at $t = 26$ in Figure 1.1. In practice, only a subset of the entire domain is queried for

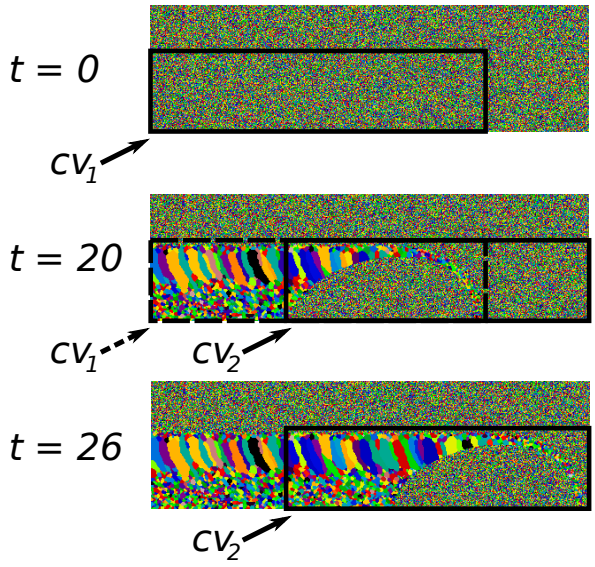


Fig. 1.1: Schematic of *stitching* example; melt pool travels to the right. At $t = 0$, the pool starts outside of the domain and we generate the first computational volume that the pool scans over. The dimensions of the computational volume encompass the *haz* in which grain evolution occurs, and only grains inside this volume are considered for computation and I/O. When the pool reaches the right boundary at $t = 20$, a second computational volume is generated and the pool continues moving until it again reaches the right boundary of the second box. By writing the time step at each output interval, *stitch* can then merge the outputs to reconstruct an image of the entire simulation domain that is spatially and temporally consistent with the progression of the simulation.

purposes of visualization and analysis.

In Section 3, we highlight ways *stitching* allows us to extend *spparks* functionality and discuss the scientific merits of these results. We further measure the execution time of *spparks* simulations and output file size with and without *stitching* to quantify the performance enhancement. Our initial studies reveal *stitch* to be a promising I/O tool that offers flexibility in data storage and has computational expense comparable to existing libraries.

2. Methods. The *stitch* database and programming API are inspired by *spparks* welding and AM applications. In this section, benefits of using the *stitch* library are summarized and briefly described; then *stitch* concepts are introduced and described relative to existing I/O approaches. Subsequently, the benefits of using *stitch* are described relative to *spparks* KMC applications of simple equiaxed grain growth and AM modeling of grain growth.

2.1. Benefits of using *stitch* I/O with *spparks*. Using the *stitch* database and library API with *spparks*, the following advantages/benefits are obtained.

1. **The *stitch* file can be accessed using a Python interface.** During *spparks* calculations of grain growth, a single *stitch* file is written using *dump stitch* for both parallel and serial calculations. A native Python interface has been developed which allows Python scripts to read/write *stitch* files using numpy arrays. This greatly enhances the ease of post-processing capabilities and facilitates image creation using existing standard Python tools.

2. **The *stitch* file can be appended and used to assemble a much larger simulation incrementally.** This *stitching* feature of a *stitch* database can result in a huge reduction in file sizes and facilitates very large simulations across multiple length scales; this allows *spparks* to incrementally append sites and associated states to an existing database, thereby extending the spatial extent of an AM simulation—much like the physical AM process of adding material. *Stitching* is further discussed and demonstrated in subsequent sections.
3. **Any rectangular sub-volume of a *stitch* file can be read for any time step.** Because *spparks* simulations can be very large, it is generally impractical and often unnecessary to read an entire *stitch* file for post processing—although that can be done if sufficient computer memory is available. On practical terms, only a sub-volume may be required for image processing and/or analysis, and the *stitch* database allows the user to query blocks of any dimension.

2.2. *Stitch* Design. The current *stitch* capability has been developed for structured grids associated with *spparks* calculations. However, *Stitch* is a significant departure from existing I/O libraries motivated in part by the AM and welding examples, but also by the understanding that there is a broader class of simulation types that can potentially benefit. For example, some finite element models have similar computational intensity in limited parts of the total simulation domain, offering an opportunity to do lossless compression by only saving parts of data that have changed.

Previous efforts to do compression on data [8,12] have focused either on looking at all of memory or by looking at the entire simulation domain and determining what could be skipped. In both cases, good "compression" is achieved, but with considerable effort. The *stitch* approach is fundamentally different and is best for simulations with isolated work properties. Other projects such as ASCR SIRIUS have investigated lossy data compression and data decomposition by incrementally loading only part of the data precision at a time, thereby progressively recreating the full precision data set. While these efforts can work more broadly, data reduction with full precision cannot approach the benefits of the *stitch* approach.

The design choices that have made *stitch* work stem from ignoring the traditional I/O library need to know what the entire simulation domain is and likewise requiring that the whole domain is written for each output. This seemingly simple shift has demonstrated radical space savings with a projected slight time advantage on radically fewer processes with no precision loss.

Internally, *stitch* can track multiple different fields, each with their own domain and with no necessary relation to each other. Each field can store scalars, vectors or tensors.

From a time perspective, typical I/O libraries rely on an integer-based time progression to match the library designer's ideas about multi-dimensional data with one dimension being time. Rather than having such a strict idea and force this view on the application developer, *stitch* uses a real value for time along with absolute and relative tolerance parameters to allow accurate matching for IEEE floating point values that notoriously cannot be compared exactly.

At an implementation level, rather than recreating much of the data selection functionality needed, an embedded relational database format is used instead. The SQLite [4] database offers a public domain code base, full ACID transactions, full SQL functionality, and the ability to share a database among a collection of different applications and processes simultaneously without introducing errors. We leverage this functionality to implement many of the different features we require. For example, to select all blocks written that intersect with a given area no later than a particular time, we have a SQL query that

takes into account the time tolerance parameters and matches the box boundaries against blocks to select matching blocks ordering from oldest to newest. This radically simplifies the manually constructed index and searching technology needed to support the *stitch* functionality. Instead of searching all of the blocks to select the correct ones, we get just the ones we need from SQLite and then copy the relevant portions into the provided buffer, *stitching* together the requested region.

The implementation of *stitch* is completely in C to facilitate easy integration into a variety of environments, and *spparks* uses the C interface directly from its C++ code base. We also provide a fully functional Python API and use the Python unit testing capabilities to test the correctness of the implementation. Given the richer array manipulation functionality Python offers compared to C, this offers very readable tests that are not stuck comparing the minutiae of array elements against a standard or even constructing such a reference array. Beyond the testing advantages, the wealth of analysis tools available in Python make this interface a crucial part of the provided functionality.

While the current code base works well for the intended purpose, there are limitations that still must be investigated and solved. First, the *stitching* procedure is correct, but not optimal. Currently it copies all data into the resulting block that would intersect with no regard for if that data is overwritten with a newer value or not. Investigations to determine if these wasted memory copy operations are significant have not been performed. Given the performance measured currently, it has not been a priority.

The second, and potentially larger, issue relates to how SQLite locks the database when writing. The current implementation is such that it locks the entire database for any writes, no matter which table is being written to. The larger issue is that it relies on POSIX file system locking to implement these locks. Parallel file systems do not always offer full POSIX support with POSIX locking being a common feature that is ignored. For example, Lustre does not properly implement POSIX locking without severe performance penalties. SpectrumScale (IBM's GPFS) seems to handle these locks correctly offering a platform for use at Sandia (gpfs scratch).

While the locking may be an issue at scale—even though we do not expect to run at scale beyond 1000 processes due to how the physics works—there are alternatives that may work. BerkeleyDB from Oracle is a drop in compile time replacement for SQLite that uses an external file to implement locks. It also has finer granularity locks, potentially prompting not just using BerkeleyDB instead of SQLite, but also database and code implementation redesign to decompose the field processing to allow greater parallelization. The challenge with BerkeleyDB is the open source license it is released under. The license is well known to be "sticky" to anything that touches it, making using alternative licenses difficult, if not impossible. This limitation prevents the initial distribution of *stitch* from using BerkeleyDB.

The code is in the final stages of copyright review and will being released under the LGPL v2.1 license shortly.

2.3. Microstructure Simulations. To demonstrate *stitch* with *spparks* and to measure performance, microstructure evolution simulations were conducted. In this subsection, two kinds of microstructure evolution and grain growth are briefly described: 1) equiaxed grain growth using the *spparks* Potts model [5], and 2) grain nucleation and evolution associated with *spparks* KMC models of AM. In both cases, *stitch* I/O is used; in the first case (Section 2.3.1), the *spparks* interface to *stitch* offers an alternative *dump stitch* command to the existing *dump text*; in the latter case (Section 2.3.2), *stitching* is used to demonstrate the ability to append to an existing database. The present section only describes methods and concepts; specific numerical results will be presented in Section 3.

2.3.1. *Stitch* I/O replacement for *spparks dump text*. The *stitch* library offers an alternative output database for the *spparks dump text* command. Rather than dumping an ordinary text file of *spparks* lattice sites and spin values, or similarly a distributed set of text files for parallel calculations, the *spparks dump stitch* command produces a single *stitch* file and database for lattice site spin values irrespective of whether or not calculations were conducted in serial or parallel. This approach introduces a new flexible way of conducting microstructure analyses and image creation through the use of a Python API to the *stitch* database file.

To assess the performance of the new *dump stitch* command, both weak and strong scaling studies were conducted. In a weak scaling study, the problem size is scaled up proportionally as additional resources (process cores) are applied to the problem; problems solved in this way are sometimes referred to as "memory bound." In a strong scaling study, the problem size is held fixed while more and more computational resources (process cores) are applied for obtaining the solution faster. Problems that are solved in this way are sometimes referred to as "CPU bound."

To measure performance, the *spparks* Potts grain growth model was used to create and evolve an equiaxed microstructure over time. Output was generated for only a few steps and regularly dumped at equal intervals. With this simple application, we only wanted to study parallel performance of reading and writing data with *stitch*, and thus the Potts model was a suitable choice. We only measure the execution time of *stitch* I/O, and do not include data post-processing (e.g. image generation). The size of the simulation domain was chosen to be on the order of 100 million total lattice sites so that performance evaluations could be completed reasonably quickly; simulation domain sizes are given in Table 3.1. *Stitch* I/O performance is measured against the following I/O methods:

- *spparks* without I/O
- *spparks* with single text file
- *spparks* with distributed text files – one file per MPI task
- *spparks* with image files (JPG or PNG files)

Spparks without I/O establishes a baseline for measuring the *stitch* I/O overhead. The single text file output by *spparks* is the most direct comparison to *stitch* as both methods write data to a single file at specified output intervals; this induces potential I/O contention between processes for both *stitch* and text I/O. The distributed text files output by *spparks* produces one text file per process per output time interval. The image I/O option generates a JPG image of a 2D slice of the simulation domain per output interval. For both strong and weak scaling studies, wall-clock time is calculated to the nanosecond in five separate iterations and averaged; the standard deviation is used for displaying error bars. Timing results are presented and discussed in Section 3.2.

2.3.2. *Stitching* an AM simulation domain: Benefits and approach. To demonstrate *stitching*, a key aspect of *stitch* functionality, *spparks* AM simulations were conducted. *Stitching* is a radical rethinking of how simulation data is written to a database. In particular, the *stitch* library and API were developed to allow appending of results to an existing database. This approach is a true digital and numerical representation of the additive manufacturing process, wherein the simulation domain is appended to at particular stages during the simulation. At no stage during the simulation is the full domain in computer memory or even specified; rather only very localized regions (called *cv* for computational volume) of the *haz* is in memory at any particular simulation time. After the *stitching* simulation is completed, the entire simulation domain is represented in the file. Any subset for any simulation time can later be extracted, analyzed and viewed.

The benefits of *stitching* were enumerated in Section 2.1. For a physical AM process,

layer-by-layer material is deposited and selectively laser melted. The moving heat source leaves behind cooling material that quickly solidifies; material microstructures in this *haz* form and evolve until the laser source has moved sufficiently far away, at which point the material will not experience further grain growth. The *stitching* paradigm handles all of these basic process conditions in an extremely performant way by facilitating reading/writing over only those active regions where grain growth and evolution are occurring.

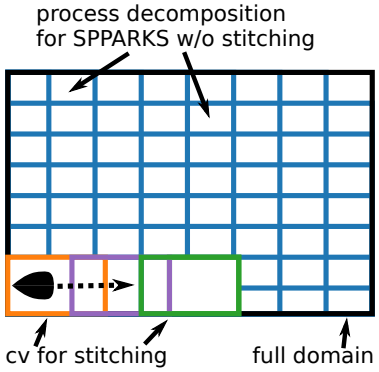


Fig. 2.1: Schematic of how *spparks* simulations differ with *stitching*. The full domain is outlined in black. Traditionally, a large simulation would require a large number of processors (64 in this example) that divide the domain into sections such that each process handles approximately the same number of sites (blue grid). The full domain is used for both computation and I/O, even though the data in a vast majority of the sites are not changing. Now, with *stitching*, the *cv* contains only the affected area as the pool moves across the domain (orange, purple, green, etc.), and only one *cv* at a time is used for computation and I/O. This makes it possible to break up the simulation over a large domain and final AM build, into a series of small simulations, each of which can be run efficiently on a much smaller number of processors.

Consequently, the *stitching* process leads to enormous reductions of required computing resources (process cores) and output file size. Figure 2.1 gives a visual interpretation for the improved distribution of computational resources in *spparks*. With the traditional approach of simulating AM on the entire build domain, the problem is memory bound; this means there is no way to store the entire simulation domain without distributing it across many processors, despite the fact that nothing is happening on most of the domain. Large domains therefore require more processors to distribute the computation. Note that in practice, because of length scale disparities between the built part and grains, it is also generally impossible to model the entire simulation domain on the largest supercomputers. In a related but more subtle way, massive reductions in output file size arise because, in contrast to writing/updating states on the entire simulation domain, *stitch* only stores states on the active *cv*; this means that for most of the domain, states are only stored for time steps when evolution is occurring and otherwise it is logically treated as a constant for later times.

For *stitching*, a Python script is used to generate the sequence of *cv*s for *spparks* that eventually get merged by *stitch*. The size of the *haz* is precomputed and used to size the *cv*, which follows the movement of the melt pool. Then, using the *stitch* library and API semantics, when computations on a new *cv* begin, states on the previous and/or overlapping *cv*s can be read in and initialized on the new *cv*. The process of *stitching* with *spparks* consists of precomputing a logical sequence of *cv*s that tile the build volume. A sequential series of

spparks simulations are performed on each *cv* and the *stitch* file grows as the simulation proceeds. Each *spparks* simulation is initialized by reading states from the growing *stitch* file. In most cases, because the domain is growing, the new *cv* partially overlaps with previously written *cv*s; states in the overlap region are initialized from the *stitch* file while new domain sites must be logically initialized in *spparks*. This process is entirely implemented as a combination of bash and Python scripts which control the workflow. Python scripts are generally used to size and position *cv*s and used to write parameter files for inclusion in *spparks* scripts. Bash scripts are generally used to loop over the logical sequence of *cv*s. We note that there is no single formula for implementing the workflow as it can be accomplished in many ways.

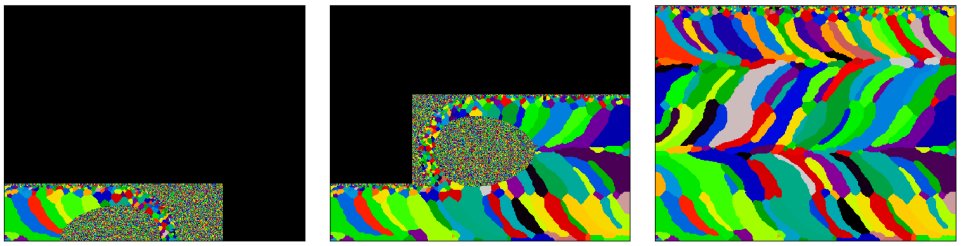
3. Results and Discussion. In this section, we present both qualitative and quantitative results of running *spparks* simulations with the *stitch* library. We first give visual examples of the metallic microstructure generated by the AM models to demonstrate the flexibility of *stitching* together *cv*s in different configurations. We then show the scaling studies for the *stitch* I/O system and how the performance compares to native *spparks* I/O. Finally, we present results on the execution time for *spparks* with and without *stitching* for domains of various sizes. Concepts and context for these simulations were described in Section 2.

3.1. Microstructure Simulations. Simulations of metallic grains were performed using *spparks* built with the C library interface for *stitch*. The code was compiled with GNU C/C++ compiler version 4.8.5 and OpenMPI 1.10. In addition, a serial Python interface of *stitch* was built using Python 3.4.6 for calculating the *stitching* domains.

Figure 3.1 depicts grain evolution during simulation of the AM weld model with *stitching*. Much like the progression of a physical AM process, only the first *cv* has to be initialized at the start of the simulation as the melt pool only affects sites along the bottom edge; the rest of the domain is assumed to be unaffected and therefore does not have to be stored in the *stitch* database. As the melt pool progresses, it moves in a serpentine pattern, reversing directions with each transverse increment, evident by the grain inclinations along the raster direction. The *cv*s are generated one at a time, and only the current one is used for computation and I/O—what is shown is a result of *stitching* together the previously computed *cv*s. Note that all sites outside of the current *haz* are constant at later time steps, which is reflected by the color of the grains in the figure. In the end, *stitch* is able to recover the full simulation domain by *stitching* together the individual *cv*s in a manner that is commensurate with the progression of the AM build.

Stitch is designed so that all *stitching* parameters are contained in a single JSON file. The JSON file is parsed using Python and generates input scripts for *spparks* on any particular *cv*. For a full description of possible inputs to the AM weld model, we point readers to the online *spparks* documentation [11]. Since each invocation of *spparks* only simulates on the *cv*, we anticipate the ability to simulate at low computational cost extremely large domains that were previously inaccessible to conventional methods. The reduced memory usage for the smaller *cv*s makes it possible to scale from laptops to clusters, depending on the simulation parameters and urgency of results. One can then use the fully-stitched domain for crystal plasticity finite element calculations [2] or data analytics studies of the microstructure. By bridging the length scale from microns to centimeters, *stitch* enables scientists to study larger systems and iterate on new experiments more efficiently than was previously possible.

3.2. *Stitch* I/O Performance. To probe the overhead of using *stitch* I/O (without *stitching*), performance evaluations were performed on the Chama capacity cluster at Sandia. It is an Infiniband-based cluster from Appro on a RHEL 7 platform with 1232 compute nodes.



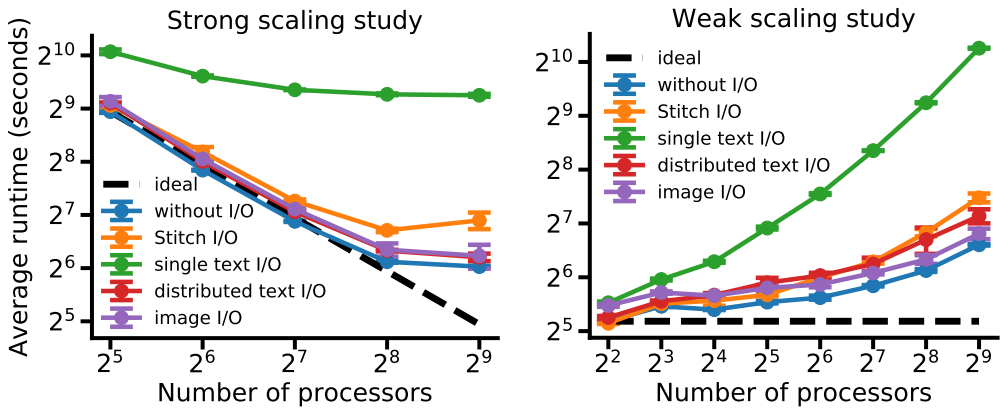
(a) $t = 26$ MCS (b) $t = 98$ MCS (c) $t = 198$ MCS

Fig. 3.1: Example of grain evolution produced by the *spparks* AM weld model with *stitching*. Note that black regions have yet to be initialized and are *not* stored in the *stitch* database. (a) Initially, only the first *cv* is initialized as the melt pool moves across the bottom edge, as the rest of the domain is assumed to be unaffected. (b) As the layers build up, the pool travels in a serpentine pattern, reversing direction with each transverse increment. Sites outside of the current *haz* are constant at later time steps, evident by the stable color of the grains. (c) In the end, *stitch* is capable of seamlessly *stitching* together the different *cv*s to recover the full simulation domain, even though the full dimensions were never explicitly recorded in the *stitch* file.

Each compute node contains two 8-core, 2.6 GHz Intel Sandy Bridge E5-2670 processors with 4 GB of RAM per core. Tests were performed using a shared 4.6 PB GPFS scratch space. We calculate the wall-clock time using five separate iterations, and report the average total execution time with error bars of one standard deviation.

Simulation domain sizes for the Potts model (see Section 2.3) were chosen to be on the order of 100 million total lattice sites so that performance evaluations could be completed reasonably quickly for a "typical-sized" problem. This led to dimensions of 1200 sites $\times$ 1200 sites $\times$ 88 sites. As seen in Figure 3.2a, we observed that *stitch* I/O performance was comparable to other forms of I/O and had nearly ideal scaling behavior up to 256 processors. For 256 processors and beyond, we observed degradation in scaling behavior for *stitch* I/O that was reflected, to varying degrees, in the other I/O methods as well. We speculate that the worsening performance and higher variability at 512 processors is due to file system contention, although a deeper analysis is required for verification. However, in all cases, we found *stitch* I/O to perform between 2 to 10 times faster than *spparks* single text I/O and achieve much better scaling; note that single file text I/O has burdensome communication issues because it requires each processor send its chunk of data to a single output processor and is hence not very scalable; this is not the case for distributed text I/O which is faster and scales better. the resulting *stitch* file size was 1.4 GB on average, which is almost 4 times smaller than the average size of the text files (5.5 GB). Thus the *stitch* file is able to store output data more compactly than the text file; furthermore, *stitch* file sizes did not vary by more than 1 MB as the number of processors was scaled up.

A weak scaling study was also conducted; in this case, the number of sites per process were kept around 5×10^5 sites/process. This led to the domain dimensions shown in Table 3.1. Figure 3.2b shows results for the same I/O formats. Again, *stitch* I/O remains competitive with other formats up to 512 processors, and still runs 2 to 10 times faster than *spparks* I/O with a single text file. Once again, the *stitch* file is 3 to 4 times smaller than the text file on average, with the ratio increasing for more processors. All of the curves deviate from ideal



(a) (b)

Fig. 3.2: Scaling performance evaluations for *spparks* with no I/O (blue), *stitch* I/O (orange), single text file I/O (green), distributed text files I/O (red), and static image (JPG) I/O (purple). The black dashed line represents ideal scaling behavior. The wall-clock time was averaged across five iterations and error bars represent ±1 standard deviation. (a) Strong scaling study for a constant domain size of 1200 × 1200 × 88. We observed relatively little impact on total execution time from *stitch* I/O up to 256 processors, and the additional execution time of *stitch* I/O is still significantly lower than that of *spparks* single text I/O. (b) Weak scaling study where domain sizes were chosen to keep the number of sites per process close to 5×10^5 sites/process. We observed good scaling of *stitch* I/O comparable to cases with static image output and distributed text output.

behavior, which suggests that the increased parallel overhead is not exclusively a *stitch* issue. This result shows that it is feasible to use *stitch* as an I/O tool on large parallel computing clusters, though we emphasize far fewer than 512 processors are required if *stitching*.

Num procs	Domain size	Num sites	Stitch file size	Text file size
4	150 × 150 × 88	1980000	23 MB	59 MB
8	216 × 216 × 88	4105728	47 MB	137 MB
16	300 × 300 × 88	7920000	91 MB	277 MB
32	426 × 426 × 88	15969888	186 MB	608 MB
64	600 × 600 × 88	31680000	363 MB	1.26 GB
128	860 × 860 × 88	65084800	753 MB	2.67 GB
256	1200 × 1200 × 88	126720000	1.42 GB	5.53 GB
512	1700 × 1700 × 88	254320000	2.91 GB	12.1 GB

Table 3.1: Parameters used in weak scaling performance study. The number of sites per process was kept constant at approximately 5×10^5 sites/process. The *stitch* file size was 3 to 4 times smaller than the size of the corresponding text file.

3.3. *Stitching* Performance. *Stitching* enables the ability to run large AM simulations on very few processors. This is in contrast to the traditional approach of running on

the final domain that may be very large; significant computational resources go to waste because the AM process is inactive on most of the domain at any particular time. *Stitching* circumvents this problem by using fewer processors and only performing computations on parts of the domain that are active; this enables AM simulations to run a desktop machine (e.g. 16 procs) rather than on a supercomputer.

Performance enhancement from *stitching* was compared to traditional *spparks* AM simulations. The parameters chosen for welding simulations were $\alpha = 0.75$, $\beta = 0.5$, $k_BT = 0.25$, *haz* width = 30 sites, melt pool speed $v_p = 10$ sites/MCS and melt pool width of 301.3 sites. These values were adapted from previous work by Rodgers et al. [17], and we point the reader there for a deeper analysis on the effect of different parameters on the weld microstructure.

For performance measurements, the domain size was progressively doubled as shown in Table 3.2. When running *spparks* without *stitching*, a weak scaling type approach was used: The number of processors was doubled while attempting to keep the number of sites per process at approximately 7031 sites/process; without *stitching*, the required number of processors and file sizes increase rapidly (see Fig. 3.3b). Thus, the domain sizes we chose were limited by the traditional approach and its computability on a reasonable number of processors. Notably, however, when simulating the same domains with *stitching*, the number of processors was kept *constant* at 16 (approx. 6230 sites/process) on the individual *cv*, which is even accessible by desktop machines. Output was written every 2 MCS to a *stitch* file and distributed text files for *stitching* and no *stitching* tests, respectively.

Num procs	Domain size	Num sites	Stitch file size	Text file size
32	750 × 300 × 1	225000	105 MB	156 MB
64	1500 × 300 × 1	450000	170 MB	663 MB
128	3000 × 300 × 1	900000	300 MB	2.68 GB
256	3000 × 600 × 1	1800000	497 MB	8.92 GB
512	3000 × 1200 × 1	3600000	890 MB	31.8 GB
1024	3000 × 2400 × 1	7200000	1.64 GB	122 GB

Table 3.2: Parameters used in the *stitching* performance study. Num procs is the number of processes for simulations without stitching. The Text file size is the sum of all distributed text files, and it is considerably larger than the size of the corresponding single *stitch* file.

Figure 3.3 compares the performance of *spparks* with and without *stitching*. We calculate the wall-clock time to the nanosecond in three separate iterations, and we report the average total execution time with error bars of one standard deviation. As shown by the orange data points, the *stitching* simulations generally took more time than the corresponding simulations without *stitching* (blue points); however, the difference is only worse by a factor of at most 3 in the worst case, even when the number of processors differed by as much as 2^6. We find the longer execution time to be a reasonable tradeoff given that we can now perform the same size simulation with far fewer computational resources. Furthermore, we observed an expected linear scaling in the *stitching* execution time, since doubling the simulation area roughly corresponds to doubling the number of *cv*s that need to be stitched together. The size of the individual *cv*s remain unchanged, which greatly reduces the memory requirements and computational burden for *spparks*.

In addition to the lower demand for processors, *stitching* provides yet another performance boost in the form of reduced storage footprint. As shown in Table 3.2 and Figure 3.3b, the *stitch* file is considerably smaller than the cumulative size of the corresponding distributed

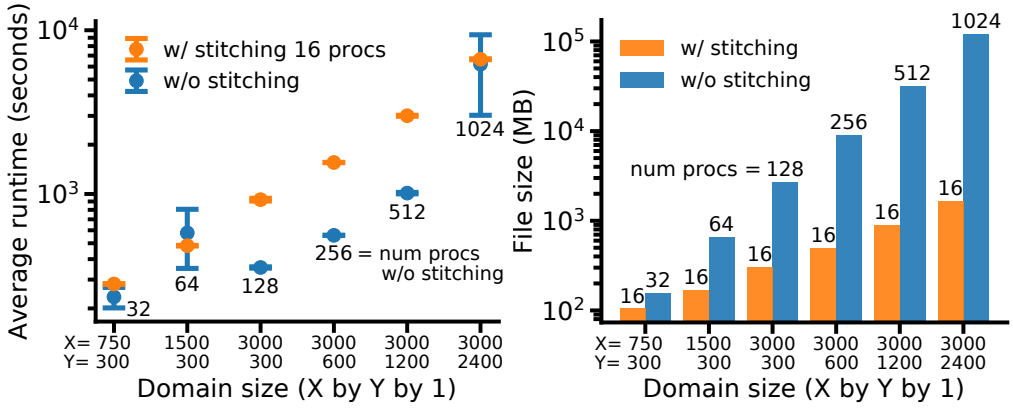


Fig. 3.3: Comparison of *spparks* performance with and without *stitching* for the AM weld model. There were approximately 7031 sites/process in the simulations without *stitching*, and the same area was used for the *stitching* simulations on just 16 processors (approx. 6230 sites/process). Each simulation was run for full coverage, and output was produced every 2 MCS. The numbers next to the data points indicate the number of processes used for that computation. (a) While *spparks* without *stitching* generally ran faster for all cases, it used many times more processors than *stitching* (up to 2^6 times more). *Stitching* has much improved scaling and runs only 2 to 3 times slower than *spparks*, despite using only 16 processors. (b) The size of the output file(s) are plotted against the domain size for both cases. The *stitch* file is 2 to 75 times smaller than the corresponding set of text files (note the log scale on the y-axis), which is a dramatic reduction in storage requirements.

text files, up to 75 times smaller for the largest problem size. The traditional approach creates massive output files because the files include data from all sites at all time steps, resulting in tremendous wasted storage space. The file size scaling without *stitching* is approximately quadratic as doubling the simulation domain not only doubles the number of sites written at each output step, but it also doubles the number of time steps that must be written; *stitching*, on the other hand, has relatively constant-sized *cvs*, and thus the *stitch* file sizes exhibit approximately linear scaling with respect to problem size.

To test the effects of problem scale, we also used the same *stitching* performance testing framework to simulate domains that were ten times thicker than the ones listed in Table 3.2 (approximately 62 300 sites/process and 70 310 sites/process for simulations with and without *stitching*, respectively). We had all simulations output data to *stitch* files to keep file sizes reasonable. The results shown in Fig. 3.4 reflect the same qualitative trends we observed in Fig. 3.3. Namely, we find that the execution time of *stitching* these larger domains is no more than twice that of a traditional *spparks* simulations, despite using the same modest quantity of 16 processors in all *stitching* runs. In addition, the output file from *stitching* is up to 9 times smaller than the file produced without *stitching*, with the factor of reduction in storage footprint only increasing as the problem size increases. The results presented here reveal how *stitching* enables *spparks* to simulate extremely large domains at low computational expense, making such AM simulations tractable on personal machines as we strive to achieve physically-relevant length scales.

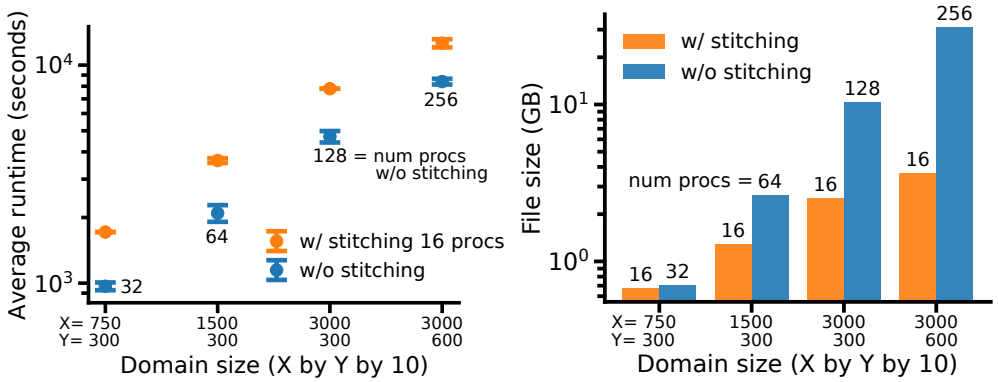


Fig. 3.4: Comparison of *spparks* performance with and without *stitching* for the AM weld model over domains 10 times larger than those in Table 3.2 (approximately 62 300 sites/process and 70 310 sites/process for simulations with and without *stitching*, respectively). *Stitching* simulations were still performed on just 16 processors, and the number of processors used is indicated next to the data. Each simulation was run for full coverage, and output was produced to a *stitch* file every 2 MCS. (a) The execution time for *stitching* is no more than twice that of a traditional *spparks* run, despite using just 16 processors for *stitching*. (b) The *stitch* files produced by *stitching* are up to 9 times smaller than *stitch* files output from the traditional approach, and *stitching* demonstrates better overall scaling with problem size.

4. Conclusion. This paper introduced the *stitch* library and associated concepts for its application to additive manufacturing simulations of grain growth using the *spparks* KMC framework. *Stitch* is a novel approach to I/O for AM simulations; it facilitates incremental growth of a database associated with a simulation, much the same way a part grows during an AM build. It was shown that not only is there no loss in I/O performance if *stitch* is used as a replacement for existing text file outputs, but *stitch* file sizes were also significantly smaller than the equivalent text file outputs used by *spparks*. The novelty of *stitch* allows for *stitching* calculations wherein only data in areas that are actively manufactured are used for computation and I/O, resulting in dramatic reductions of computational resources and file sizes. Demonstration calculations showed the same calculation could be conducted by *stitching* with only 16 processors compared to 1024 processors using the traditional approach. *Stitching* adds flexibility by enabling out-of-core computation and makes formerly unreachable AM simulations on physically-relevant length scales more computationally feasible. Future work will be directed at extending these results further into true 3D space, and integrating the C library API with more applications such as finite element codes for thermal calculations of AM thermal processes. As scientists continue to push the size limits of computer simulations, we anticipate a growing demand for the functionality demonstrated by the *stitch* library for AM and other use cases.

Acknowledgments. We thank Steve Plimpton for guidance on integrating *stitch* into *spparks* and helpful scaling study discussions. We also thank Stewart Silling and Veena Tikare for support and helpful discussions related to this work. We are grateful to the following Sandia programs for funding this work: 1) ASC Physics and Engineering Models

(P&EM) Advanced Manufacturing, Materials Performance and Solid Mechanics, 2) Integrated Modeling and Applications, and 3) Advanced Certification Program. This work was also supported in part under the U.S. Department of Energy National Nuclear Security Agency ATDM project funding and the U.S. Department of Energy Office of Science, under the SSIO grant series, SIRIUS project and the Data Management grant series, Decaf project, program manager Lucy Nowell.

REFERENCES

[1] J. A. Elliott, *Novel approaches to multiscale modelling in materials science*, International Materials Reviews, 56 (2011), pp. 207–225.
[2] M. M. Francois et al, *Modeling of additive manufacturing processes for metals: Challenges and opportunities*, Current Opinion in Solid State and Materials Science, 21 (2017).
[3] W. E. Frazier, *Metal additive manufacturing: A review*, Journal of Materials Engineering and Performance, 23 (2014), pp. 1917–1928.
[4] D. R. Hipp, *SQLite*. https://www.sqlite.org/index.html, 2000.
[5] E. A. Holm and C. C. Battaile, *The computer simulation of microstructural evolution*, JOM, 53 (2001), pp. 20–23.
[6] M. Horstemeyer, *Integrated Computational Materials Engineering (ICME) for Metals: Using Multiscale Modeling to Invigorate Engineering Design with Science*, Wiley, 2012.
[7] Lockheed Martin, *Giant satellite fuel tank sets new record for 3-d printed space parts*. https://news.lockheedmartin.com/2018-07-11-Giant-Satellite-Fuel-Tank-Sets-New-Record-for-3-D-Printed-Space-Parts, 2018. Accessed: 2018-08-16.
[8] J. Lofstead, G. Jean-Baptiste, and R. Oldfield, *Delta: Data reduction for integrated application workflows and data storage*, in High Performance Computing, M. Taufer, B. Mohr, and J. M. Kunkel, eds., Cham, 2016, Springer International Publishing, pp. 142–152.
[9] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, *Adaptable, metadata rich IO methods for portable high performance IO*, in 2009 IEEE International Symposium on Parallel Distributed Processing, 2009, pp. 1–10.
[10] J. H. Martin, B. D. Yahata, J. M. Hundley, J. A. Mayer, T. A. Schaedler, and T. M. Pollock, *3D printing of high-strength aluminum alloys*, Nature, 549 (2017), pp. 365–369.
[11] J. A. Mitchell and S. Plimpton. http://spparks.sandia.gov/doc/app_potts_additive.html.
[12] B. Nicolae and F. Cappello, *AI-Ckpt: Leveraging memory access patterns for adaptive asynchronous incremental checkpointing*, in Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing, HPDC '13, New York, NY, USA, 2013, ACM, pp. 155–166.
[13] S. Plimpton, A. Thompson, and A. Slepoy, *SPPARKS*. http://spparks.sandia.gov/index.html, 2009.
[14] S. Plimpton et al, *Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo*, Tech. Rep. SAND2009-6226, Sandia National Laboratories, 2009.
[15] R. K. Rew and G. P. Davis, *NetCDF: An interface for scientific data access*, IEEE Computer Graphics and Applications, 10 (1990), pp. 76–82.
[16] T. M. Rodgers, J. D. Madison, and V. Tikare, *Simulation of metal additive manufacturing microstructures using kinetic Monte Carlo*, Computational Materials Science, 135 (2017), pp. 78–89.
[17] T. M. Rodgers, J. A. Mitchell, and V. Tikare, *A Monte Carlo model for 3D grain evolution during welding*, Modelling and Simulation in Materials Science and Engineering, 25 (2017), p. 064006.
[18] The HDF Group, *Hierarchical Data Format, version 5*, 1997-2018. http://www.hdfgroup.org/HDF5/.
[19] A. F. Voter, *Introduction to the kinetic Monte Carlo method*, in Radiation Effects in Solids, K. E. Sickafus, E. A. Kotomin, and B. P. Uberuaga, eds., Dordrecht, 2007, Springer Netherlands, pp. 1–23.
[20] S. Yip and M. P. Short, *Multiscale materials modeling at the mesoscale*, Nature Materials, 12 (2013), pp. 774–777.

Computational Mathematics

Computational mathematics is concerned with the design, analysis, and implementation of algorithms to solve mathematical problems. Articles in this section describe the discretization of partial differential equations and methods to solve equations, couple multiphysics systems of equations, and quantify uncertainty.

Holtzer, Bochev, and Peterson study feature-preserving optimization methods. They apply optimization-based property-preserving methods to advection-diffusion problems, assess them numerically, and compare them to algebraic flux correction schemes.

Levitt, Boman, Rajamanickam, and Biros study an algebraic method to compress the Green's function in electromagnetic scattering simulations. They extend the geometry-oblivious fast multipole method to support nonsymmetric complex matrices and assess its ability to accurately compress the Green's function.

Bonilla, Mabuza, Shadid, and Badia present a stabilized continuous finite element scheme for first order conservation laws in scalar transport problems based on an artificial diffusion operator and a differentiable, linearity preserving shock detector. They assess their method for scalar convection and present a preliminary solution for a steady Euler equation problem.

Marvin, Wildey, and Bui-Thanh develop a scalable approach to the solution of linear and nonlinear stochastic inverse problems using the recently developed consistent Bayesian method. They demonstrate the scalability of their approach on an advection-diffusion equation and a hyperelasticity equation.

Robertson, Khalil, and Adams study the challenge of Markov chain Monte Carlo (MCMC) sampling. They compare several tools available for general MCMC samplers to determine recommendations for diagnostic implementation in the Dakota software.

Reeder, Hill, Aimone, and Severa extend a neural algorithm for solving the diffusion equation PDE by implementing random walks on neuromorphic hardware.

Ohm, Berger-Vergiat, and Tuminaro study strategies to exploit the mesh structure for better performance. They develop an aggregation based multigrid method that preserves local structure for applications where complex unstructured meshes are only required in small sub-regions of the overall domain.

Nelsen and Bosler introduce a meshless Lagrangian particle-based numerical method to solve the shallow water equations for ocean modeling. They study challenging physical boundaries such as a moving wet/dry line.

Bogle, Devine, Perego, Rajamanickam, and Slota study degenerate mesh features in climate simulations of ice-sheets which pose problems for solvers. They present a parallelizable breadth-first-search-based label-propagation approach which removes all degenerate features during each step of an ice-sheet simulation efficiently.

Schrum, Jr., Rintoul, and Newton discuss the detection of anomalous trajectories. They introduce a semantic categorization into trajectory analysis by including additional parameters such as trajectory curvature, distance, and total deflection.

Smith, Kaushagen, Hoemmen, and Siefert address the challenge of parameters selection for convergent solutions of partial differential equations. They present a machine-learning model which provides linear solver and preconditioner settings.

A. Cangi
M. L. Parks

December 6, 2018

APPLICATIONS OF OPTIMIZATION-BASED TRANSPORT

NIKKI HOLTZER*, PAVEL BOCHEV†, AND KARA PETERSON‡

Abstract. Unlike structure-preserving methods, feature-preserving discretization methods are still relatively underdeveloped. In this report we consider optimization-based property-preserving methods for advection-diffusion problems motivated by similar techniques for remap between two different meshes. The work studies these methods numerically and compares them to algebraic flux correction schemes.

1. Introduction. The discretization of models which have physical properties that one would like to include in the modeling process must be treated with care. Failure to do so, can amount to discrete equations that are not well posed or that provide unphysical solutions. For problems which are collections of physical models, the situation can become increasingly worse. The unphysical solution may cause a trickle down effect. In such cases, solutions may obey basic mathematical conditions such as consistency and stability, however still be next to useless physically.

Although, research in structure-preserving discretization methods is extensive; see, e.g., [1,4] and the references therein, limited work exists in feature-preserving discretization methods. In such discretization methods, qualitative features of the exact solution are sought to be reproduced. Unfortunately, structure preserving methods are not guaranteed to also preserve features such as local solution bounds, maximum principle, and positivity, to name a few. This paper seeks to use an optimization approach first applied in the context of remap [3] to capture features whilst formulating stable and consistent solutions for a model advection-diffusion equation. In particular, here we consider a problem configuration involving an anisotropic diffusion coefficient motivated by Magnetohydrodynamic (MHD) applications.

This paper is organized as follows. Section 2 describes an abstract optimization-based setting for property preserving discretizations and its specializations to remap and advection-diffusion problems. Sections 3 will describe the model anisotropic diffusion problem and provide preliminary numerical results. Section 4 will outline ideas for future work.

2. An abstract property-preserving scheme. We consider an optimization-based property-preserving approach comprising the following two key steps:

1. Use any standard discretization scheme to compute a formally accurate but not necessarily property preserving *target solution*.
2. Minimize the mismatch between the discrete solution we seek and the target solution, subject to constraints enforcing the desired physical properties.

This approach has the following advantages [2]:

1. Given the target solution, the solution found with the additional constraints is always the best possible, with respect to the selected measure of mismatch.
2. Accuracy is separated from the preservation of the properties.
3. Feature-preserving methods can be utilized on unstructured meshes.
4. Due to the fact that the only requirements for the scheme defining the target are stability and consistency, adaptive procedures can be used where the solution oscillates. If the solution or its derivatives have discontinuities across a cell, the target can be altered via the use of discontinuous spaces.

A formal description of the approach follows. Suppose we wish to find a solution $u \in X$

*University of Arizona, Applied Mathematics, nholtzer@math.arizona.edu
†Center for Computing research, Sandia National Laboratories, pbboche@sandia.gov
‡Center for Computing research, Sandia National Laboratories, kjpeter@sandia.gov

such that

$$L(u) = f \in Y$$

where X, Y are Banach spaces, $f \in Y$ is given, and the operator $L : X \to Y$. We assume there exists finite dimensional subsets $\mathcal{U} \subseteq X$ and $\mathcal{V} \subseteq Y$ such that for *any* $f \in \mathcal{V}$ the solution, $u \in \mathcal{U}$. These subsets encode the desired solution properties. Also, we assume there exists discrete spaces, X_h and Y_h, and discrete operators, $L_h : X_h \to Y_h$, such that there there exists a unique solution to the following problem:

$$L_h(u_h) = f_h \quad \forall h.$$

Furthermore, if $f_h \to f$ then $u_h \to u$. For simplicity, we consider a conforming approximation setting where

1. $X_h \subset X$ and $Y_h \subset Y$;
2. $\mathcal{U}_h = \mathcal{U} \bigcap X_h$ and $\mathcal{V}_h = \mathcal{V} \bigcap Y_h$.

Note we have *not* assumed that the discrete solution is property preserving, i.e., that the following holds:

- $f_h \in \mathcal{V}_h$ and $u_h \in \mathcal{U}_h$.

Instead, we introduce an optimization target solution $u_h^T := u_h$ and seek a *property-preserving* discrete solution $\hat{u}_h$ such that $\hat{u}_h \in \mathcal{U}_h$ and $f_h \in \mathcal{V}_h$. To find such a solution we treat the physical properties as constraints and consider the following optimization problem:

$$\text{minimize} \;\; J(\hat{u}_h; u_h^T) := \frac{1}{2}||\hat{u}_h - u_h^T||_W^2 \;\; \text{subject to} \;\; \begin{cases} L_h(u_h^T) = f_h \\ \hat{u}_h \in \mathcal{U}_h \\ L_h(\hat{u}_h) \in \mathcal{V}_h. \end{cases}$$

The paper [3] specializes this approach to the constrained interpolation (remap) problem, while [2] further extends it to a semi-Lagrangian transport scheme. This report considers application of the abstract approach to an Eulerian finite element method for the advection-diffusion equation. The preserved features will be both conservation of mass and the density.

For completeness we review specialization of the approach to remap and then consider advection diffusion problems. This requires some additional notation. Let Ω be a polyhedral domain in $\mathbb{R}^d$. Assume that $C(\Omega)$ is a given *source* mesh on Ω and $\tilde{C}(\Omega)$ is another, *target* mesh on the same domain. We denote the source mesh cells by c_i and their vertices by v_i. The set of all source mesh vertices is denoted by $V(\Omega)$. The corresponding entities on the target mesh will be denoted with a tilde. We make the following assumptions regarding the meshes:

1. The vertices which define $\partial\Omega$ belong to both $C(\Omega)$ and $\tilde{C}(\Omega)$.
2. If $\tilde{v}_i$ is on the boundary then v_i is on the boundary also and both belong to the same part of the boundary.

We denote the space of piecewise constant functions on $C(\Omega)$ by C^h and let C_0^h be its zero-mean subspace, i.e.,

$$C_0^h = \left\{ u \in C^h \mid \sum_{i=1}^{N_C} u_i = 0 \right\},$$

where N_C is the number of mesh cells. The analogues of these spaces on $\tilde{C}(\Omega)$ are denoted by a tilde. We endow all discrete spaces with the l_2 inner product.

2.1. Optimization-based finite element transport (OBFET). We first show how the abstract property preserving framework specializes to the mass-density remap problem [2].

Let ρ be a positive scalar function (density) and

$$M = \int_{\Omega} \rho(\mathbf{x}) dV$$

denote the total mass in Ω. The cell mass and the mean cell density on the source mesh are given by

$$m_i = \int_{c_i} \rho(\mathbf{x}) dV \quad \text{and} \quad \rho_i = \frac{m_i}{\mu(c_i)}, \tag{2.1}$$

respectively, where $\mu(c_i)$ is the measure of c_i. It follows that $m_i = \rho_i \mu(c_i)$ and $M = \sum_{i=1}^{N_C} \rho_i \mu(c_i)$

The mean density satisfies the following local bounds

$$\rho_i^{min} \leq \rho_i \leq \rho_i^{max} \quad \forall i = 1...N_C$$

where min and max are taken over the nearest neighbors of cell c_i. It follows that

$$m_i^{min} = \rho_i^{min} \mu(c_i) \leq m_i \leq \rho_i^{max} \mu(c_i) = m_i^{max} \quad \forall i = 1...N_C.$$

For the mass-density remap we assume that ρ_i is given on every cell c_i of the source mesh and $\rho(\mathbf{x})$ is given on the boundary. We wish to find approximation of the cell masses on the target mesh $\tilde{C}(\Omega)$, i.e.,

$$\tilde{m}_i \approx \int_{\tilde{c}_i} \rho(\mathbf{x}) dV \,.$$

We require the following conditions be satisfied:

1. $\sum_{i=1}^{N_{\tilde{C}}} \tilde{m}_i = \sum_{i=1}^{N_C} m_i = M$ (Mass Conservation)
2. $\tilde{m}_i = \int_{\tilde{c}_i} \rho(\mathbf{x}) dV$, $\forall \rho(\mathbf{x}) = a_0 + a_1 \mathbf{x}$. (Linear Density Preservation)
3. $\tilde{m}_i^{min} = \rho_i^{min} \tilde{\mu}_i \leq \tilde{m}_i \leq \rho_i^{max} \tilde{\mu}_i = \tilde{m}_i^{max}$ (Local Bound Preservation)

We specialize the abstract framework to this problem as follows. Let $\rho(\mathbf{x}) \in R$ and $L : R \to \tilde{C}^h$ be the operator which computes the exact cell masses on $\tilde{C}$, i.e.,

$$(\tilde{m})_i = (L(\rho(\mathbf{x})))_i := \int_{\tilde{c}_i} \rho(\mathbf{x}) dV.$$

Since L is not known in general, we replace it by an approximation $L_h : C^h \times C^h \to \tilde{C}^h$ which operates on the data that is given, and is exact for density functions belonging in some set $\mathcal{B}$, that is,

$$\tilde{m} = L_h(m, u(\rho)) \quad \forall \rho(\mathbf{x}) \in \mathcal{B}$$

However, the range of L_h is **not** required to satisfy conditions (1) and (3) from above. Here we assume that $\mathcal{B}$ is the class of linear density functions, i.e. $\rho(\mathbf{x}) = a_0 + a_1 \mathbf{x}$. To state the optimization problem we view the above problem as an equation defining the optimization target:

$$L_h(m, u^T) = \tilde{m} \,.$$

The abstract optimization formulation then yields the following property-preserving remap scheme:

$$\min \frac{1}{2}||\hat{u} - u^T||_{l_2}^2 \quad \text{subject to} \quad \begin{cases} L_h(m, u^T) = \tilde{m} \ \ \forall \rho(\mathbf{x}) \in \mathcal{B} \\ \sum\limits_{i=1}^{N_C} m_i = \sum\limits_{i=1}^{N_{\tilde{C}}} (L_h(m, \hat{u}))_i \\ \tilde{m}^{min} \leq L_h(m, \hat{u}) \leq \tilde{m}^{min} \end{cases} \tag{2.2}$$

We refer to [2,3] for further details and specific variations of this formulation which differ in the manner in which the targets are defined. We next provide a brief summary of a flux-based remap since a similar setting will be utilized for the advection-diffusion problem.

2.2. Flux-corrected remap (FCR). The optimization-based remap in Section 2.1 is related to a flux-corrected remap (FCR) scheme [8] motivated by the classical Algebraic Flux Correction (AFC) approach [5]. In this section we briefly review FCR. To that end it we retain the following notation

$$m_i = \int\limits_{c_i} \rho(\mathbf{x}) dV; \quad \rho_i = \frac{m_i}{\mu(c_i)}; \quad \text{and} \quad M = \int\limits_{\Omega} \rho(\mathbf{x}) dV = \sum_{i=1}^{C} m_i = \sum_{i=1}^{C} \rho_i \mu(c_i). \tag{2.3}$$

from the mass-density case and adopt the necessary additional notation from [8,9].

As before, we seek an accurate approximation of the mass on $\tilde{C}(\Omega)$ such that mass is conserved and the remapped density satisfies local bounds. The FCR scheme uses the so-called flux representation of the density on the new mesh, which is given by

$$\tilde{m}_i = m_i + \sum_k F_{i,k}^m \tag{2.4}$$

where $F_{i,k}^m = -F_{k,i}^m$ are mass fluxes defined as

$$F_{i,k}^m \approx \int\limits_{\tilde{C}_i \cap C_k} \rho dV - \int\limits_{C_i \cap \tilde{C}_k} \rho dV \, ; \tag{2.5}$$

see [8]. The antisymmetry of the fluxes ensures that mass is conserved. Following [8] we rewrite the local bounds in terms of the flux:

$$m_i^{min} \leq m_i + \sum_k F_{i,k}^m \leq m_i^{max} \tag{2.6}$$

Accuracy of the approximation depends on the accuracy of the flux. Typically, a *high-order* flux, denoted by $F_{i,k}^{m,H}$, is obtained by using a linear reconstruction of the density in (2.5). However, the high-order flux is not guaranteed to satisfy the local bounds (2.6). Thus, the goal is to find fluxes $F_{i,k}^m$, which satisfy (2.6) and are close to the high order fluxes $F_{i,k}^{m,H}$. In [8] this task is formulated as a global optimization problem:

$$F^m = \arg\min \|F^m - F^{m,H}\|_{l^2}^2 \quad \text{subject to (2.6)}. \tag{2.7}$$

The FCR scheme results from simplifying the constraints in (2.7) to a set of box constraints which decouple the variables and allow to find an approximate solution by solving small local problems. This process assumes that there exist low-order fluxes $F_{i,k}^{m,L}$ satisfying these bounds, so that the candidate solution can be written as

$$F_{i,k}^m = F_{i,k}^{m,L} + C_{i,k}^m dF_{i,k}^m \quad \text{where} \quad dF_{i,k}^m = -dF_{k,i}^m = (F_{i,k}^{m,H} - F_{i,k}^{m,L}) \, , \tag{2.8}$$

and $0 \le C^m_{i,k} = C^m_{k,i} \le 1$ are some yet to be determined coefficients. We now wish to find $C^m_{i,k}$ such that (i) the bounds (2.6) hold and (ii) $C^m_{i,k}$ are as close as possible to 1. One can show that an admissible set of coefficients is given by

$$C^m_{i,k} = \begin{cases} \min(D^{m,+}_i, D^{m,-}_k, 1) & \text{if } dF^m_{i,k} > 0 \\ \min(D^{m,-}_i, D^{m,+}_k, 1) & \text{if } dF^m_{i,k} < 0 \\ 1 & \text{if } dF^m_{i,k} = 0 \end{cases} \tag{2.9}$$

where

$$D^{m,+}_i = \frac{Q^{m,max}_i}{P^{m,+}_i}\,; \quad D^{m,-}_i = \frac{Q^{m,min}_i}{P^{m,-}_i}\,;$$

$$Q^{m,max}_i = m^{max}_i - \tilde{m}^L_i \ge 0\,, \quad Q^{m,min}_i = m^{min}_i - \tilde{m}^L_i \le 0$$

and

$$P^{m,+}_i = \sum_{dF^m_{i,k}>0} dF^m_{i,k} \ge 0\,, \quad P^{m,-}_i = \sum_{dF^m_{i,k}<0} dF^m_{i,k} \le 0.$$

see [8] for further details.

3. Model Problem. We consider the advection-diffusion equation

$$\frac{\partial \varphi}{\partial t} + \nabla \cdot (-\mathcal{D}\nabla\varphi + u\varphi) = f \text{ on } \Omega \times [0,T] \tag{3.1}$$

augmented with the boundary and initial conditions

$$\varphi = \varphi_D \text{ on } \partial\Omega \times [0,T] \quad \text{and} \quad \varphi(\mathbf{x},0) = \varphi_0, \tag{3.2}$$

respectively. In (3.1) the domain Ω is a bounded open region in $\mathbf{R}^2$, $\mathcal{D}$ is a prescribed diffusion coefficient, and u is a prescribed velocity field. We assume a problem setting such that the solution φ preserves the mass in Ω, i.e.,

$$\int_\Omega \varphi(x,t)dx = \int_\Omega \varphi(x,0)dx = \mu_0 \quad \forall 0 < t \le T\,. \tag{3.3}$$

3.1. Extension of OBFET and FCR to the model problem. Assume that $C(\Omega)$ is a finite element partition of Ω and (3.1) is discretized in space by a standard C^0 nodal Galerkin scheme using piecewise linear or bilinear elements. We assume that time discretization is by an implicit or explicit one step scheme and denote the corresponding fully discrete equation giving the finite element solution at the new time step as

$$L_h(\varphi^h) = f^h\,.$$

For simplicity we omit the index of the current time step.

Let φ^{min}_i and φ^{max}_i denote the minimum and maximum nodal solution values at the current time step of the nodes adjacent to node v_i. We seek a discrete solution at the new time step which satisfies the local bounds

$$\varphi^{min}_i \le \varphi_i \le \varphi^{max}_i \quad \forall\, i = 1...N_V, \tag{3.4}$$

where N_V is the number of vertices in the mesh $C(\Omega)$. To obtain such a solution we specialize the OBFET formulation (2.2) as follows. We define the target as the solution of the Galerkin problem and then seek a property-preserving finite element field by minimizing the L^2 norm of the difference between this field and the target. It is easy to see that a discrete version of property (3.3) is given by

$$\mathbf{1}^T \mathbb{M} \boldsymbol{\varphi} = \mu_0$$

where $\mathbb{M}$ is a consistent mass matrix, $\mathbf{1}$ is a vector of ones and $\boldsymbol{\varphi}$ is the coefficient vector of the finite element solution φ^h at the current time step. The optimization-based property-preserving formulation of (3.1) is then given by

$$\min \frac{1}{2}||\varphi^h - \varphi^T||^2_{0,\Omega} \quad \text{subject to} \quad \begin{cases} L_h(\varphi^T) = f^h \\ \mathbf{1}^T \mathbb{M} \boldsymbol{\varphi}^h = \mu_0 \\ \varphi_i^{min} \le \varphi_i \le \varphi_i^{max} \quad \forall i = 1...N_V. \end{cases} \tag{3.5}$$

The second property-preserving formulation for (3.1) is based on a representation of the nodal values of the solution φ^h in terms of nodal fluxes. Specifically, we have the following analogue of (2.3):

$$(\boldsymbol{\varphi})_i = (\boldsymbol{\varphi}_L)_i + \sum_{j \neq i} \omega_{ij} F_{i,j}$$

where $F_{i,j}$ is a nodal flux from node v_j into node v_i and ω_{ij} is a weight which includes an orientation factor, the lumped nodal mass and the time step; see, e.g., [7]. This representation requires a notion of a low-order finite element solution φ_L^h which satisfies the local solution bounds (3.4) and preserves the mass, i.e.,

$$\mathbf{1}^T \mathbb{M} \boldsymbol{\varphi}_L = \mu_0 \, .$$

Under these assumptions one can show that with slight modifications to account for the fact that now $F_{i,j}$ represent nodal fluxes, formula (2.9) provides a solution in the present case as well. This type of property-preserving methods is often referred to as algebraic flux correction, or AFC schemes; see [6, 7].

3.2. Computational study. Simulations of the Magnetohydrodrynamic (MHD) equations have expanded our knowledge of magnetic fields. Further understanding of magnetic fields is essential to the field of astrophysics. This is due primarily to the sheer amount of magnetized plasma existing in the universe. These fields can provide us information of forces, stresses, and even transport in plasma. They allow diffusion along field lines while keeping transport across them at bay. Anisotropic transport, specifically, affects many fundamental properties within a plasma. Unfortunately, research of anisotropic transport in plasmas is still insufficient [10]. As a preliminary step towards better understanding of the underlying processes, here we present numerical results for an anisotropic diffusion version of the model problem (3.1) that provides a simplified model of the relevant physics.

3.3. Anisotropic Diffusion. For clarity of physical meaning we restate the model problem (3.1) as

$$\frac{\partial T}{\partial t} + \nabla \cdot (\mathcal{D} \nabla T - v T) = f \text{ on } \Omega \times [0, \tau] \tag{3.6}$$

augmented with the boundary and initial conditions

$$T = T_D \text{ on } \partial\Omega \times [0,\tau] \quad \text{and} \quad T(\mathbf{x},0) = T_0, \tag{3.7}$$

respectively. Here, T is a temperature profile, $\Omega := [-2,2]\times[-2,2]$, $\mathcal{D} = \{d_{ij}\}$ is a prescribed anisotropic diffusion tensor coefficient, and v is a prescribed velocity field. We note that $\mathcal{D}$ is made up of diffusivity parameters in the parallel, perpendicular, and angular direction of the flow, each of which scales like $||B||^{-i}$ where $i = 1, 2$ and B is a given magnetic field. $\mathcal{D}$ will be some tensor which will be cylindrical in nature and T_0 is an initial temperature profile that we will evolve in time.

3.4. Numerical Results. We compare property-preserving solutions of the anisotropic diffusion example (3.6) using the Optimization-based Finite Element Transport (OBFET) scheme (3.5) and the algebraic flux correction (AFC) method based on the flux representation of the finite element solution. In the numerical experiments that follow, we have set $v = 0$ and are using a forward Euler discretization scheme. Moreover, the initial temperature profile, T_0, is taken to be a Gaussian distribution on a ring; see Fig. 3.1. We define our magnetic field, which we evaluate at integration points x and y, in the following way:

$$B = \left\{ \tfrac{-B_0 x}{r}, \quad \tfrac{B_0 y}{r} \right\}$$

where

1. $B_0 = 1$
2. $r = \sqrt{x^2 + y^2}$.

Furthermore, the diffusion coefficient $\mathcal{D}$ is defined below.
If $B > 0$, we have,

$$\mathcal{D}\nabla T = \kappa_{||}\nabla_{||} T + \kappa_{\perp}\nabla_{\perp} T.$$

Otherwise, we have,

$$\mathcal{D} = \kappa_{||}$$

where for the previous two definitions:

$$\kappa_{||} = 1, \ \kappa_{\perp} = 0$$

and finally $\nabla_{||(\perp)} T$ denotes the projection of ∇T onto the parallel (perpendicular) direction of the magnetic field. We use free boundary data on all boundaries. Our principal goal is to investigate how these schemes handle the high material contrast in the example problem.

Figure 3.1 shows the initial temperature profile. Then we display the solution plots, utilizing the previously presented AFC scheme, as well as x and y solution slices, for a 64x64 mesh respectively.

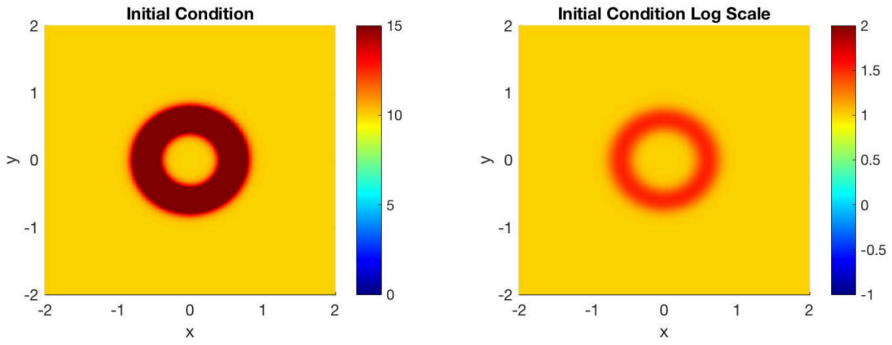


Fig. 3.1: Initial temperature profile T_0.

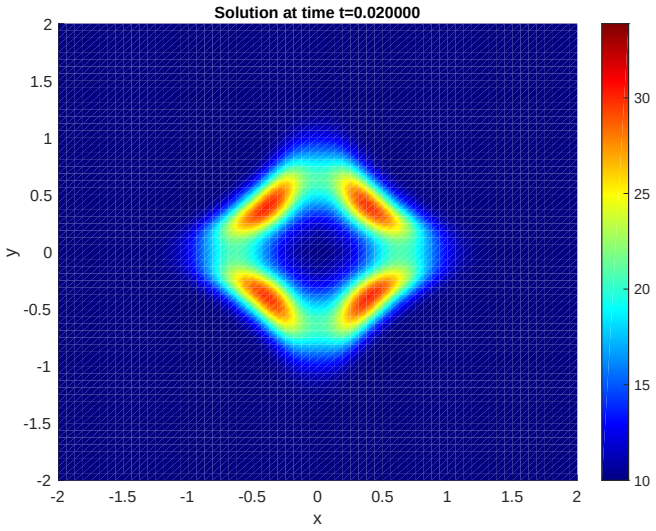


Fig. 3.2: AFC solution on 64x64 mesh

We now display the same plots corresponding to one variation in the optimization based remap approach, with a 64x64 mesh.

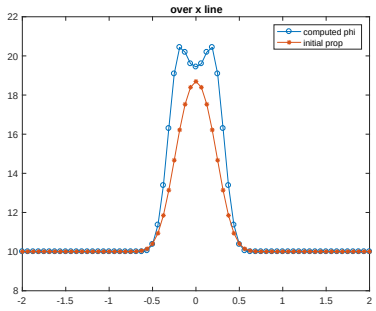


Fig. 3.3: y solution slice: AFC on a 64x64 mesh.

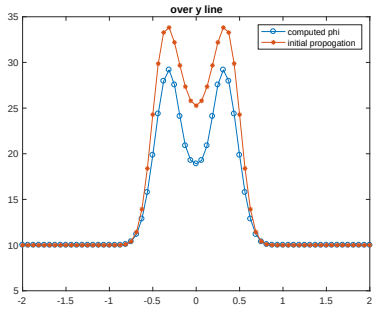


Fig. 3.4: x solution slice: AFC on a 64x64 mesh.

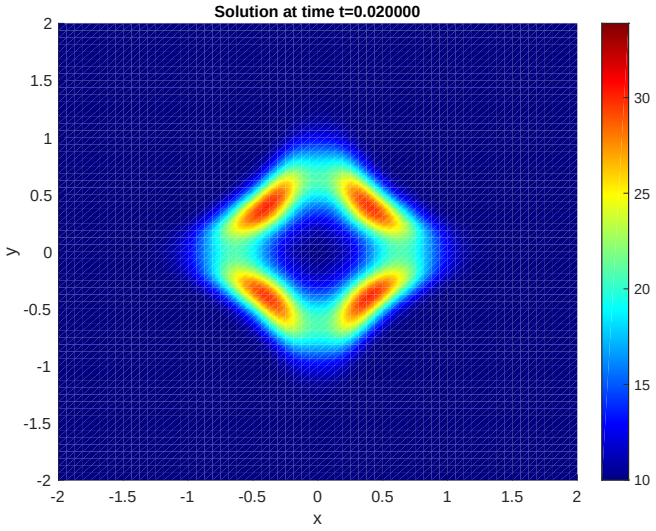


Fig. 3.5: OBFET solution on 64x64 mesh

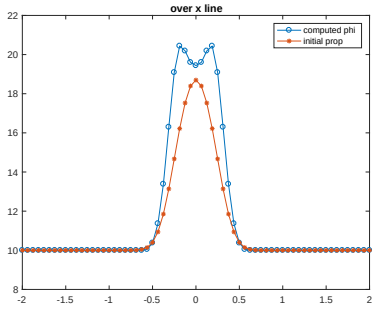


Fig. 3.6: y solution slice: OBFET on 64x64 mesh.

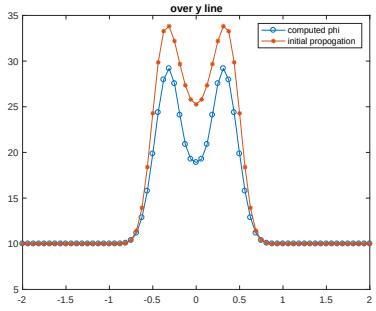


Fig. 3.7: x solution slice: OBFET on 64x64 mesh.

For comparison, we increase our mesh to 128x128 and display the solution plots for both AFC and OBFET and their corresponding x and y slices below.

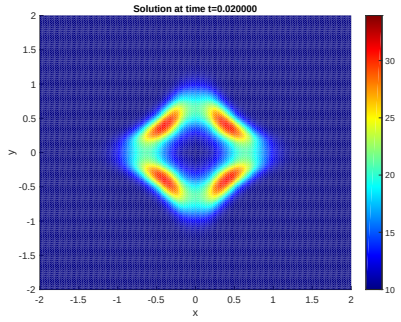


Fig. 3.8: AFC

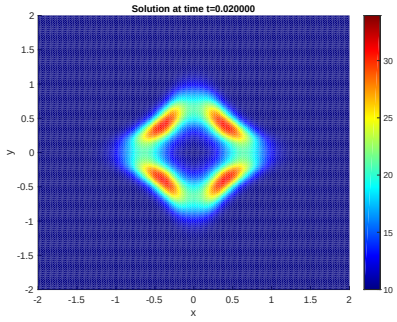


Fig. 3.9: OBFET

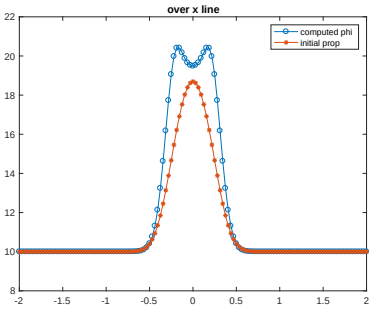


Fig. 3.10: AFC

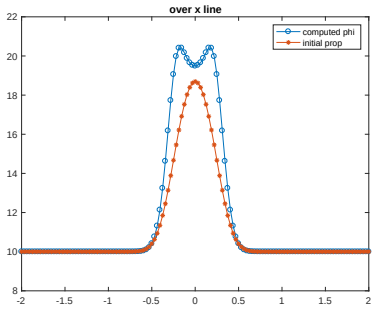


Fig. 3.11: OBFET

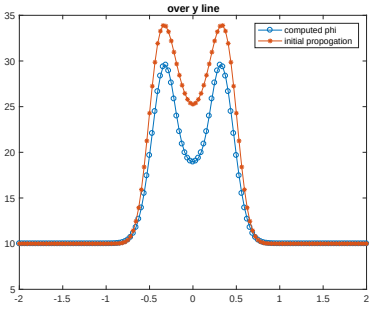


Fig. 3.12: AFC

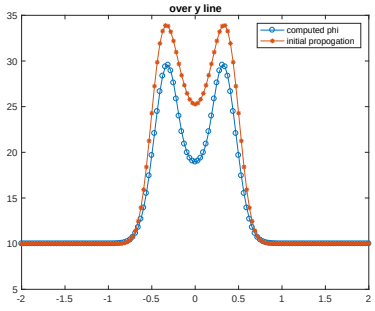


Fig. 3.13: OBFET

We see similar behavior between the two methods with the refined mesh, as expected. We note that the solution still continues to have oscillations, a problem we will address in the final section of this paper.

4. Future Work. We begin by noting that our work with anisotropic diffusion, utilizing AFC, didn't exactly match previously obtained results. In the future, we will try utilizing a

radially discretized mesh i.e. each element will be a chunk of an annulus. This will provide a more realistic representation of transport since the field lines will no longer be aligned with the mesh. This should also improve our OBFET solutions as well. Moreover, we wish to run all simulations with a more refined mesh since the above plots utilize a relatively coarse mesh.

We aim to finish implementing a mixed finite element scheme. In doing so, we will implement both simplicial and quad Raviart-Thomas elements. Then, we plan to combine the optimization-based approach with a mixed finite element discretization of the model problem. Such a combination has not yet been investigated and it presents several challenges. Specifically, a mixed formulation involves a direct approximation of the flux and the scalar variable. It is an open question whether or not enforcing bounds on the scalar alone will ensure an acceptable flux approximation. If bounds for the flux are also required, the structure of the optimization problem may change, potentially requiring a new optimization algorithm.

REFERENCES

[1] D. N. Arnold, R. S. Falk, and R. Winther, *Finite element exterior calculus, homological techniques, and applications*, Acta Numerica, 15 (2006), pp. 1–155.

[2] P. Bochev, D. Ridzal, and K. Peterson, *Optimization-based remap and transport: A divide and conquer strategy for feature-preserving discretizations*, Journal of Computational Physics, 257 (2014), pp. 1113–1139.

[3] P. Bochev, D. Ridzal, G. Scovazzi, and M. Shashkov, *Formulation, analysis and numerical study of an optimization-based conservative interpolation (remap) of scalar fields for arbitrary Lagrangian-Eulerian methods*, Journal of Computational Physics, 230 (2011), pp. 5199 – 5225.

[4] P. B. Bochev and J. M. Hyman, *Principles of mimetic discretizations of differential operators*, in Compatible Spatial Discretizations, D. N. Arnold, P. B. Bochev, R. B. Lehoucq, R. A. Nicolaides, and M. Shashkov, eds., vol. 142 of The IMA Volumes in Mathematics and its Applications, Springer New York, 2006, pp. 89–119.

[5] J. P. Boris and D. L. Book, *Flux-corrected transport. i. shasta, a fluid transport algorithm that works*, Journal of Computational Physics, 11 (1973), pp. 38 – 69.

[6] D. Kuzmin, *Algebraic Flux Correction I*, Springer Netherlands, Dordrecht, 2012, pp. 145–192.

[7] D. Kuzmin, R. Löhner, and S. Turek, eds., *Flux-Corrected Transport. Principles, Algorithms and Applications*, Springer Verlag, Berlin, Heidelberg, 2005.

[8] R. Liska, M. Shashkov, P. Váchal, and B. Wendroff, *Optimization-based synchronized flux-corrected conservative interpolations (remapping) of mass and momentum for arbitrary lagrangian-eulerian methods*, Journal of Computational Physics, 229 (2010), pp. 1467–1497.

[9] R. Liska, M. Shashkov, P. Vachal, and B. Wendroff, *Synchronized flux corrected remapping for ale methods*, Computers & Fluids, 46 (2011), pp. 312 – 317. 10th ICFD Conference Series on Numerical Methods for Fluid Dynamics (ICFD 2010).

[10] B. Vaidya, D. Pasad, A. Mignone, P. Sharma, and L. Ricker, *Scalable explicit implementation of anisotropic diffusion with runge-kutta-legendre super-time-stepping*, arXiv:1702.05487, (2017).

NONSYMMETRIC ALGEBRAIC FMM WITH APPLICATION TO COMBINED FIELD INTEGRAL EQUATIONS

JAMES L. LEVITT*, ERIK G. BOMAN†, SIVASANKARAN RAJAMANICKAM‡, AND GEORGE BIROS§

Abstract. Boundary integral equation methods are well-established tools for simulating electromagnetic scattering. One of their shortcomings is that, upon discretization, they result in a linear system with a *dense matrix* K. This problem is well understood and has been addressed with Fast Multipole Methods (FMM), which date back to the early 90s. By introducing a controllable approximation error, such methods turn an $O(N^2)$ matrix-vector product with K to an $O(N)$ matrix-vector product under suitable conditions. Classical FMM schemes are based on analytic expansion of the underlying Green's function and they essentially sparsify or *compress* K. From a software implementation point of view, they are rather intrusive and require careful integration with the discretization code. This makes their use a bit difficult, especially if one wants to experiment with different discretization schemes for the boundary integral equation. An alternative is to use algebraic methods to compress K, the so-called *H-matrix* methods.

Here we test one such method, the geometry-aware variant of GOFMM, on a low-frequency scattering problem discretized using the combined field integral formulation and the method of moments (essentially a Petrov-Galerkin collocation scheme). This scheme gives rise to a non-Hermitian matrix. Current H-matrix methods, including GOFMM, require row and column permutations of the matrix in order to compress K and importance sampling to construct low-rank approximations. To construct these permutations and sample rows and columns we need to know the coordinates of the collocation points. If K were SPD, we could apply the geometry-oblivious variant of GOFMM, which does not require such coordinates. But unfortunately K is not symmetric. In this work we extend the geometry-aware variant of GOFMM to support nonsymmetric complex matrices and test it on a test matrix generated by the Eiger library developed at Sandia National Laboratories. Since geometric information was unavailable, we use the natural ordering of the matrix (no permutations) and uniform random sampling. Under these conditions, the Eiger matrix K compresses well up to a relative error of 3E-4, which is insufficient for some right-hand sides since the matrix has condition number of 7E-8. Further compression of K may be possible with a better permutation and higher quality sampling that leverage geometric information. To demonstrate the possibility of attaining arbitrary accuracies when the coordinates of the collocation points are available, we also test GOFMM on a synthetic problem that results in a non-symmetric complex matrix related to low-frequency scattering from point scatterers in three dimensions.

1. Introduction. Given a perfect electric conductor and an incident electric field, we seek the induced current on the surface of the conductor due to the incident field, which can then be used to find the scattered electric field. The induced current is governed by the combined field integral equation (CFIE) [12], which is a weighted sum of the electric field integral equation (EFIE) and magnetic field integral equation (MFIE). The problem can be solved with either the EFIE or MFIE separately, and while they produce symmetric moment matrices, such approaches encounter problems with homogeneous solutions corresponding to interior resonance entering into the solution. Use of the CFIE avoids such issues at the cost of losing symmetry of the discretized integral equation operator K.

The problem is discretized with the method of moments using Rao-Wilton-Glisson (RWG) basis functions [13] to yield a linear system $Kx = b$, where $K \in \mathbb{C}^{n\times n}$ and $x, b \in \mathbb{C}^n$. The discretized problem is generated using Eiger [6,8], a parallel code for simulating frequency-domain electromagnetic scattering. There are a number of concerns in formulating the problem relating to the discretization, efficient computation of moment matrix entries, and avoiding singularities, all of which are handled in Eiger and not discussed in this paper.

The matrix K is dense and the system is typically solved using a Krylov method such as GMRES [14], in which the main computational cost is in applying K to a vector. In

*The University of Texas at Austin, jlevitt@ices.utexas.edu
†Sandia National Laboratories, egboman@sandia.gov
‡Sandia National Laboratories, srajama@sandia.gov
§The University of Texas at Austin, biros@ices.utexas.edu,

order to accelerate the algorithm, we construct a hierarchical low-rank approximation (a *compression*) $\tilde{K}$ of K such that $\|\tilde{K} - K\| < \|K\|$. The matrix compression algorithm is an algebraic variant of the Fast Multipole Method (FMM) which takes advantage of hierarchical low-rank structure in the matrix to construct a compressed representation of the matrix with $\mathcal{O}(n \log n)$ work that can be applied to a vector with $\mathcal{O}(n)$ work under suitable conditions on the problem. For certain problems, the compressed matrix-vector product delivers significant speedup over the $\mathcal{O}(n^2)$ uncompressed dense matrix-vector product, leading to a faster overall runtime, even accounting for compression time.

There are a number of related works that use FMM-like algorithms for the fast solution of electromagnetic scattering problems in the engineering literature. Many methods [4,15–17] have appeared that aim at accelerating matrix-vector products. For a review of nearly optimal and highly accurate methods for frequency-domain FMM methods, see [3]. For a state of the art discretization of integral equations for electromagnetics, see [1].

As discussed, there are many efficient techniques for accelerating matrix-vector products with K. However, the majority of existing methods require tight integration of the discretization and kernel evaluation codes with the acceleration scheme. Few schemes can be used in black-box fashion and, to our knowledge, no existing software scales to distributed memory architectures. In this project, we aim at exploring black-box schemes that scale on distributed memory architectures. There are numerous works in the literature relating to hierarchical low-rank approximation of rank-structured matrices, and so we only mention those that are most closely related. We implement our scheme using the Geometry-Oblivious Fast Multipole Method (GOFMM) library [18,19]. One of the main contributions of GOFMM is the generalization of the ASKIT FMM [10] to general symmetric positive-definite matrices that do not necessarily represent interactions between pairs of points. However, we do not attempt to generalize the geometry-oblivious features of GOFMM for two reasons: the problem setting of this work is geometry-aware, and there is no guarantee of symmetry and positive-definiteness of K, a requirement for the geometry-oblivious features. Instead, we extend the geometry-aware FMM, as implemented in the GOFMM library, to nonsymmetric matrices.

2. Hierarchical Iterative Solver. For matrix $K \in \mathbb{C}^{n \times n}$ and right hand side $b \in \mathbb{C}^n$, in order to solve the linear system $Kx = b$, our first goal is to construct a hierarchical matrix approximation $\tilde{K}$ that satisfies $\|\tilde{K} - K\| < \epsilon_c$, where ϵ_c is a user-defined absolute error tolerance for the compressed operator. This compressed operator is then used to efficiently compute matrix-vector products inside an iterative solver.

Depending on the matrix and error tolerance, such an approximation may not exist Applicability of this approach requires a matrix that can be permuted to reveal hierarchical low-rank structure and an appropriate choice of error tolerance. Depending on the matrix, an error tolerance that is too small results in very little compression, leading to poor performance.

We begin this section with the hierarchical clustering of points, which defines a reordering of the matrix that seeks to reveal low-rank structure in the off-diagonal blocks. Next, we review the Interpolative Decomposition (ID), the low-rank matrix factorization that we use to approximate off-diagonal blocks. We then describe the two main phases of the algorithm, compression and evaluation, and follow with a comparison with the symmetric algorithm of [18]. Finally, we discuss the use of the compressed operator in accelerating GMRES.

2.1. Clustering/reordering. The compressibility of matrix K depends on a proper ordering of points $\{x_i\}$. Therefore, we begin by reindexing the points (reordering the matrix) such that points that are nearby in space have nearby indices. This is done by creating a hierarchical clustering of the points represented by a balanced binary tree. The root node of

the tree is assigned the full set of points. The set of points assigned to a node is split into two balanced clusters, and each cluster is assigned to a child node. This process is continued at each level of the tree until the number of points assigned to the leaf nodes fall within some prescribed maximum leaf size. We split sets of points by their projections along an axis which heuristically estimates the axis along which the points have maximal variation. This method scales well with the dimensionality d of the data [11]. The hierarchical clustering algorithm is outlined in Algorithm 4.

Algorithm 4 Construct a cluster tree for points $\{x_i\}$

procedure CLUSTER_TREE($\{x_i\}$)
  α =NEW_NODE($\{x_i\}$)
  if $|\{x_i\}| >$ `max_leaf_size` **then**
    $\bar{x} = \frac{1}{|\{x_i\}|} \sum x_i$ ▷ Compute the centroid
    $x_p = \arg\max_i \|x_i - \bar{x}\|$ ▷ Find the point farthest from $\bar{x}$
    $x_q = \arg\max_i \|x_i - x_p\|$ ▷ Find the point farthest from x_p
    `med` = MEDIAN($\{(x_i, x_p - x_q)\}_i$) ▷ Split points by projections along $x_p - x_q$
    α`.left_child` = CLUSTER_TREE($x_i : (x_i, x_p - x_q) \leq$ `med`)
    α`.right_child` = CLUSTER_TREE($x_i : (x_i, x_p - x_q) >$ `med`)
  end if
  return α
end procedure

2.2. Interpolative Decomposition. Consider the task of constructing a low rank approximation of $K_{\alpha\beta} \in \mathbb{C}^{n_\alpha \times n_\beta}$, the block of K corresponding to interactions between tree nodes α and β.

A rank-r *column ID* of a matrix $K_{\alpha\beta}$ takes the form

$$K_{\alpha\beta} \approx K_{\alpha\beta} S^c_{\alpha\beta} P^c_{\alpha\beta}, \tag{2.1}$$

where $S^c_{\alpha\beta} \in \mathbb{R}^{n_\beta \times r}$ is a column submatrix of a permutation matrix and $P^c_{\alpha\beta} \in \mathbb{C}^{r \times n_\beta}$. The product $K_{\alpha\beta} S^c_{\alpha\beta}$ is a column submatrix of $K_{\alpha\beta}$, and matrix $S^c_{\alpha\beta}$ may be viewed as selecting a subset of the columns of $K_{\alpha\beta}$ to use as a basis. The selected columns are referred to as *skeleton columns*. Then the entire decomposition may be viewed as an expansion of each column of $K_{\alpha\beta}$ in the column basis $K_{\alpha\beta} S^c_{\alpha\beta}$, where matrix $P^c_{\alpha\beta}$ encodes the expansion coefficients. Definitions of the ID in other works often use two terms rather than three, combining the product $K_{\alpha\beta} S^c_{\alpha\beta}$ into a single term. Writing the decomposition with three terms is equivalent and leads to clearer exposition in following sections.

Similarly, a rank-r *row ID* of $K_{\alpha\beta}$ takes the form

$$K_{\alpha\beta} \approx P^r_{\alpha\beta} S^r_{\alpha\beta} K_{\alpha\beta}, \tag{2.2}$$

where $S^r_{\alpha\beta} \in \mathbb{R}^{r \times n_\alpha}$ is a row submatrix of a permutation matrix and $P^r_{\alpha\beta} \in \mathbb{C}^{n_\alpha \times r}$.

A column ID and row ID of $K_{\alpha\beta}$ may be combined to form a *two-sided ID* by a simple substitution:

$$K_{\alpha\beta} \approx K_{\alpha\beta} S^c_{\alpha\beta} P^c_{\alpha\beta} \approx P^r_{\alpha\beta} S^r_{\alpha\beta} K_{\alpha\beta} S^c_{\alpha\beta} P^c_{\alpha\beta} \tag{2.3}$$

Storage of this decomposition only requires $2rn$ values for $P^c_{\alpha\beta}$ and $P^r_{\alpha\beta}$, and $2r$ indices to encode the columns/rows selected by $S^c_{\alpha\beta}$ and $S^r_{\alpha\beta}$.

The algorithm for constructing an ID is given in Algorithm 5. Given some overall error tolerance ϵ_c, the goal is to construct an ID for submatrix $K_{\alpha\beta}$ with an adaptively-chosen rank r that is less than or equal to the maximum allowed rank s such that the approximation satisfies some accuracy requirement. If such an approximation cannot be found, the matrix block is deemed incompressible. Entries belonging to an incompressible block may still be approximated if they also belong to a different block that is compressible. Otherwise, they are computed exactly. The majority of the computational cost comes from a rank-revealing QR factorization (RRQR) and a triangular solve. In order to reduce the computational cost, we use a sampled ID, in which only a small subset of the columns or rows are used in computing the results S, P [11]. The sample submatrix is selected using using neighbor-based importance sampling [9].

Algorithm 5 Construct a sampled Interpolative Decomposition of matrix $K_{\alpha\beta}$

procedure COLUMN_ID($K_{\alpha\beta}$)
  $\gamma =$IMPORTANCE_SAMPLE(α)
  $Q, R, \Pi =$RRQR($K_{\alpha\gamma}$)
  $r = \min\left(\{i : \|R[i:, i:]\| < \frac{\sqrt{|\alpha||\gamma|}}{n}\epsilon_c\}\right)$
  if $r > s$ **then**
    return `fail`
  end if
  $S = \Pi[:, :r]$
  $R_{11} = R[:r, :r]$
  $R_{12} = R[:r, r:]$
  $P = \begin{bmatrix} I & R_{11}^{-1}R_{12} \end{bmatrix} \Pi^{-1}$
  return S, P
end procedure
procedure ROW_ID($K_{\alpha\beta}$)
  $S, P =$ COLUMN_ID($K_{\alpha\beta}^T$)
  return S^T, P^T
end procedure

2.3. Compression. Constructing interpolative decompositions independently for each block $K_{\alpha\beta}$ would be inefficient and would lead to inefficiency in the evaluation phase. Instead, we construct these approximations using a recursive approach based on the cluster tree. For each tree node β, we define the *column off-diagonal block of* β to be submatrix $K_{\bar{\beta}\beta}$, and the *row off-diagonal block of* β to be submatrix $K_{\beta\bar{\beta}}$, where $\bar{\beta} = \{1, \ldots, n\} \setminus \beta$ denotes the set of points not in β. During compression, we construct approximations for these column and row off-diagonal blocks. These approximations are later used during evaluation to approximate a compressible block $K_{\alpha\beta}$ with a two-sided ID using the row ID of $K_{\alpha\bar{\alpha}}$ and the column ID of $K_{\bar{\beta}\beta}$.

For a leaf node β, we define the *column skeletonization* of β to be a low-rank approximation of $\bar{\beta}$ using the ID

$$K_{\bar{\beta}\beta} \approx K_{\bar{\beta}\beta} S_{\bar{\beta}\beta} P_{\bar{\beta}\beta}. \tag{2.4}$$

We define the *row skeletonization* of β similarly:

$$K_{\beta\bar{\beta}} \approx P_{\beta\bar{\beta}} S_{\beta\bar{\beta}} K_{\beta\bar{\beta}}. \tag{2.5}$$

For an internal node α, we only compute a column ID using columns that are skeletons of the children of α, That is, a column ID is computed for the submatrix $K_{\bar{\alpha}\alpha'}$, where $\alpha' \subset \alpha$ denotes the set of skeleton columns of the child nodes $\mathtt{l}, \mathtt{r}$ of α:

$$K_{\bar{\alpha}\alpha'} \approx K_{\bar{\alpha}\alpha'} S_{\bar{\alpha}\alpha'} P_{\bar{\alpha}\alpha'}, \qquad \text{where } K_{\bar{\alpha}\alpha'} = K_{\bar{\alpha}\alpha} \begin{bmatrix} S_{\bar{\mathtt{l}}\mathtt{l}} & \\ & S_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix}. \tag{2.6}$$

This will be combined with the IDs of the children of α to implicitly define an ID for the entire off-diagonal block of α. The skeletons satisfy the *nesting property*: the skeleton columns of α are a subset of the skeleton columns of its children. The nesting property is important for achieving $\mathcal{O}(n)$ work complexity in the evaluation stage.

Due to the nesting property, we can use (2.6) together with the column IDs of the child nodes to implicitly define a column ID of the full block $K_{\bar{\alpha}\alpha}$:

$$\begin{aligned} K_{\bar{\alpha}\alpha} &\approx K_{\bar{\alpha}\alpha} \begin{bmatrix} S_{\bar{\mathtt{l}}\mathtt{l}} & \\ & S_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} \begin{bmatrix} P_{\bar{\mathtt{l}}\mathtt{l}} & \\ & P_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} \\ &\approx K_{\bar{\alpha}\alpha'} \begin{bmatrix} P_{\bar{\mathtt{l}}\mathtt{l}} & \\ & P_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} \\ &\approx K_{\bar{\alpha}\alpha'} S_{\bar{\alpha}\alpha'} P_{\bar{\alpha}\alpha'} \begin{bmatrix} P_{\bar{\mathtt{l}}\mathtt{l}} & \\ & P_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} \\ &\approx K_{\bar{\alpha}\alpha} \begin{bmatrix} S_{\bar{\mathtt{l}}\mathtt{l}} & \\ & S_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} S_{\bar{\alpha}\alpha'} P_{\bar{\alpha}\alpha'} \begin{bmatrix} P_{\bar{\mathtt{l}}\mathtt{l}} & \\ & P_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} \end{aligned}$$

This defines a column ID of the full off-diagonal block $K_{\bar{\alpha}\alpha} = S_{\bar{\alpha}\alpha} P_{\bar{\alpha}\alpha}$ with telescoping expressions for $S_{\bar{\alpha}\alpha}$ and $P_{\bar{\alpha}\alpha}$:

$$S_{\bar{\alpha}\alpha} = \begin{bmatrix} S_{\bar{\mathtt{l}}\mathtt{l}} & \\ & S_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} S_{\bar{\alpha}\alpha'}, \qquad P_{\bar{\alpha}\alpha} = P_{\bar{\alpha}\alpha'} \begin{bmatrix} P_{\bar{\mathtt{l}}\mathtt{l}} & \\ & P_{\bar{\mathtt{r}}\mathtt{r}} \end{bmatrix} \tag{2.7}$$

We never explicitly form $P_{\bar{\alpha}\alpha}$, but instead use the telescoping expression to apply $P_{\bar{\alpha}\alpha}$ to a vector during evaluation. The row skeletonization of interior node α is carried out analogously, using the row skeletons of the children of α.

Compression consists of skeletonizing each node in the cluster tree. The tree nodes are skeletonized following a post-order tree traversal in order to satisfy data dependencies. The compression algorithm is outlined in Algorithm 6.

2.4. Evaluation. Once the compressed representation has been constructed, we seek to apply the compressed matrix to *weight vector* w to compute the *product vector* $u \approx Kw$. We refer to this step as *evaluation*. Suppose the block $K_{\alpha\beta}$ may be approximated with a two-sided ID as in (2.3) using the skeletonizations constructed in §2.3. We seek to apply it to the appropriate subvector w_β and add the contribution $K_{\alpha\beta} w_\beta$ to u_α. This is done in three steps: we first compute the *skeleton weights* $\tilde{w}_\beta = P_{\bar{\beta}\beta} w_\beta$, then the *skeleton products* $\tilde{u}_\alpha = S_{\alpha\bar{\alpha}} K_{\alpha\beta} S_{\bar{\beta}\beta} \tilde{w}_\beta$, and finally add to the products $u_\alpha \mathrel{+}= P_{\alpha\bar{\alpha}} \tilde{u}_\alpha$. Computing the skeleton products consists of a single matrix-vector product using the submatrix of $K_{\alpha\beta}$ selected by $S_{\alpha\bar{\alpha}}$ and $S_{\bar{\beta}\beta}$, rather than separately applying the three matrices.

We make a few observations that lead to an efficient implementation. First, the matrix blocks whose columns are indexed by β all belong to the column off-diagonal block of β, so they share the same skeleton weights $P_{\bar{\beta}\beta} w_\beta$. Similarly, the matrix blocks whose rows are indexed by α all apply the same matrix $P_{\alpha\bar{\alpha}}$ to their skeleton products, so we may sum their skeleton products before applying $P_{\alpha\bar{\alpha}}$ and apply $P_{\alpha\bar{\alpha}}$ a single time to the summed skeleton products. With this perspective, we define the skeleton weights of node β to be the skeleton

Algorithm 6 Compress the matrix by skeletonizing each tree node

procedure COMPRESS()
  SKELETONIZE(`root`)
end procedure
procedure SKELETONIZE(α)
  if IS_LEAF_NODE(α) **then**
    $S_{\bar{\alpha}\alpha}, P_{\bar{\alpha}\alpha} = \text{COLUMN\_ID}(K_{\bar{\alpha}\alpha})$
    $S_{\alpha\bar{\alpha}}, P_{\alpha\bar{\alpha}} = \text{ROW\_ID}(K_{\alpha\bar{\alpha}})$
  else
    SKELETONIZE(α`.left_child`)
    SKELETONIZE(α`.right_child`)
    $\alpha' = \alpha$`.left_child.skeleton` $\cup\, \alpha$`.right_child.skeleton`
    $S_{\bar{\alpha}\alpha'}, P_{\bar{\alpha}\alpha'} = \text{COLUMN\_ID}(K_{\bar{\alpha}\alpha'})$
    $S_{\alpha'\bar{\alpha}}, P_{\alpha'\bar{\alpha}} = \text{ROW\_ID}(K_{\alpha'\bar{\alpha}})$
  end if
end procedure

weights associated with all sub-blocks of its column off-diagonal blocks, and the skeleton products of α to be the skeleton products summed over the sub-blocks of its row off-diagonal block.

We also observe that as a consequence of the nesting property and telescoping expression (2.7), there is significant overlap in the computation of an interior node's skeleton products and the computation of its children's skeleton products. That is, once the skeleton products of the children are known, the skeleton products of the parent may be computed with a single additional matrix-vector product. Similarly, in computing products, rather than applying the full telescoping expression to its skeleton products, a parent node α may apply a single matrix-vector product and "pass down" the partial result $P_{\alpha'\bar{\alpha}}\tilde{u}_\alpha$ to its children, which add the appropriate parts to their skeleton products.

The evaluation algorithm is outlined in Algorithm 7. In order to satisfy data dependencies, the computation of skeleton weights is structured as a post-order tree traversal, the computation of skeleton products is structured as a visitation of tree nodes in any order, and the computation of products is structured as a pre-order tree traversal. Under suitable assumptions on the low-rank structure of K, the computational complexity of evaluation is $\mathcal{O}(n)$. Detailed cost breakdown along with some other discussion can be found in [18].

2.5. Accelerated GMRES.

2.5.1. Analysis. To accelerate GMRES, we construct a compressed operator $\tilde{K}$ and substitute it for the uncompressed operator K throughout GMRES. This leads to two different kinds of residual: we define the *relative residual of x in the compressed operator* to be $\|\tilde{K}x - b\|/\|b\|$ and the *relative residual of x in the uncompressed operator* or *true residual* to be $\|Kx - b\|/\|b\|$. These two residuals may be different, and in the accelerated algorithm, only the residual in the compressed operator is available. The following theorem gives upper bounds for the solution error and the true residual in terms of the residual in the compressed operator, the error in the compression, and the condition number of K.

THEOREM 2.1. *For matrix K and right hand side b, let x be the solution of $Kx = b$, and $\tilde{x}$ be an approximate solution of $\tilde{K}\tilde{x} \approx b$ with residual $r = \tilde{K}\tilde{x} - b$. Assume K is nonsingular,*

Algorithm 7 Apply the compressed matrix to a vector w.

procedure EVALUATE(w)
  COMPUTE_SKELETON_WEIGHTS(`root`)
  COMPUTE_SKELETON_PRODUCTS()
  COMPUTE_PRODUCTS(`root`)
end procedure
procedure COMPUTE_SKELETON_WEIGHTS(α)
  if IS_LEAF_NODE(α) **then**
    $\tilde{w}_\alpha = P_{\tilde{\alpha}\alpha} w_\alpha$
  else
    $\mathtt{l} = \alpha$`.left_child`
    $\mathtt{r} = \alpha$`.right_child`
    COMPUTE_SKELETON_WEIGHTS(`l`)
    COMPUTE_SKELETON_WEIGHTS(`r`)
    $\tilde{w}_\alpha = P_{\tilde{\alpha}\alpha'} \begin{bmatrix} \tilde{w}_\mathtt{l} \\ \tilde{w}_\mathtt{r} \end{bmatrix}$
  end if
end procedure
procedure COMPUTE_SKELETON_PRODUCTS()
  for $\alpha \in$ `tree` **do**
    $\tilde{u}_\alpha = \sum_{\beta \in \alpha\mathtt{.interaction\_list}} S_{\alpha\tilde{\alpha}} K_{\alpha\beta} S_{\tilde{\beta}\beta} \tilde{w}_\beta$
  end for
end procedure
procedure COMPUTE_PRODUCTS(α)
  if IS_LEAF_NODE(α) **then**
    $u_\alpha = P_{\alpha\tilde{\alpha}} \tilde{u}_\alpha$
  else
    $\mathtt{l} = \alpha$`.left_child`
    $\mathtt{r} = \alpha$`.right_child`
    $\begin{bmatrix} \tilde{w}_\mathtt{l} \\ \tilde{w}_\mathtt{r} \end{bmatrix} \mathrel{+}= P_{\alpha'\tilde{\alpha}} \tilde{u}_\alpha$
    COMPUTE_PRODUCTS(`l`)
    COMPUTE_PRODUCTS(`r`)
  end if
end procedure

$\|\tilde{K} - K\| < \epsilon_c \|K\|$, $\|r\| < \epsilon_r \|b\|$, *and* $\epsilon_c \, \mathit{cond}(K) < 1$. *Then*

$$\frac{\|\tilde{x} - x\|}{\|x\|} < \frac{(\epsilon_r + \epsilon_c)\, \mathit{cond}(K)}{1 - \epsilon_c \mathit{cond}(K)}$$

and

$$\frac{\|K\tilde{x} - b\|}{\|b\|} < \epsilon_r + \epsilon_c \mathit{cond}(K) \frac{1 + \epsilon_r \mathit{cond}(K)}{1 - \epsilon_c \mathit{cond}(K)}.$$

Proof. First, we bound $\|\tilde{x}\|/\|x\|$. Define $\delta K = \tilde{K} - K$. It follows from the definitions that

$$(K + \delta K)\, \tilde{x} = b + \delta b$$

$$\left(I + K^{-1}\delta K\right)\tilde{x} = x + K^{-1}r$$
$$\tilde{x} = \left(I + K^{-1}\delta K\right)^{-1}\left(x + K^{-1}r\right)$$

Matrix $I + K^{-1}\delta K$ is invertible since $\|K^{-1}\delta K\| < \epsilon_c \texttt{cond}(K) < 1$. Taking norms and using the Neumann series bound of $\left(I + K^{-1}\delta K\right)^{-1}$,

$$\|\tilde{x}\| \leq \frac{\|x\| + \|K^{-1}r\|}{1 - \|K^{-1}\delta K\|}$$
$$\frac{\|\tilde{x}\|}{\|x\|} \leq \frac{1 + \epsilon_r \texttt{cond}(K)}{1 - \epsilon_c \texttt{cond}(K)} \tag{2.8}$$

To bound $\|\tilde{x} - x\|/\|x\|$, we observe from the definitions that $\tilde{x} - x = K^{-1}r - K^{-1}\delta K\tilde{x}$. Taking norms,

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \frac{\|K^{-1}r\|}{\|x\|} + \frac{\|K^{-1}\delta K\tilde{x}\|}{\|x\|}$$
$$\leq \epsilon_r \texttt{cond}(K) + \epsilon_c \texttt{cond}(K)\frac{\|\tilde{x}\|}{\|x\|}$$

Combining this with 2.8 yields the first inequality.

To bound $\|K\tilde{x} - b\|/\|b\|$, we observe from the definitions that $K\tilde{x} = r - \delta K\tilde{x}$. Taking norms,

$$\frac{\|K\tilde{x} - b\|}{\|b\|} \leq \frac{\|r\|}{\|b\|} + \frac{\|\delta K\tilde{x}\|}{\|b\|}$$
$$\leq \epsilon_r + \epsilon_c \frac{\|K\|\|x\|}{\|b\|}\frac{\|\tilde{x}\|}{\|x\|}$$
$$< \epsilon_r + \epsilon_c \texttt{cond}(K)\frac{\|\tilde{x}\|}{\|x\|}.$$

Combining this with 2.8 yields the second inequality.
□

2.5.2. Trilinos Implementation. Trilinos [5] is a large open-source software project consisting of a collection of packages for solving large-scale numerical problems arising in computational science and engineering. The Belos package [2] provides a number of iterative solvers, including an implementation of GMRES, which we combine with the hierarchical compression algorithm to create an accelerated linear solver. Trilinos is designed to support modularity of components via the use of abstract interfaces. In particular, the GMRES implementation operates on abstract classes for operator and multivector types. Swapping out different implementations of the abstract classes does not require any changes to the solver, and the user is free to implement any operator and multivector types that satisfy the abstract interfaces. The operator interface is defined in the Belos source file `BelosOperator.hpp` and the multivector interface in `BelosMultiVec.hpp`. Simple unoptimized example implementations are located in `MVOPTester/` in the Belos test directory.

We implement a new operator to support hierarchical approximation of operator K. The new operator type creates a compressed representation of a given matrix when its constructor is called. Subsequent calls to apply the operator execute the compressed matrix-vector product of Algorithm 7.

Compared to the reference algorithm that uses an uncompressed matrix-vector product, the accelerated algorithm has an additional startup cost associated with compression. In

order to achieve speedup in total runtime, the savings due to the compressed matrix-vector product must overcome the cost of compression. For suitable problems, the compressed matrix-vector product is asymptotically faster, even accounting for compression, and therefore will be faster given a large enough problem. We empirically compare the accuracy and convergence of the accelerated algorithm against the reference algorithm in §3.

3. Experiments. The compression and evaluation algorithms of §2 were implemented in the GOFMM codebase and integrated with the GMRES solver of Trilinos. Experiments were conducted on a server equipped with dual Intel Xeon Platinum 8160 CPUs for a total of 48 physical cores and 192 GB of memory. Though the GOFMM library includes support for distributed-memory parallelism, we run these experiments only on a single compute node, using OpenMP for shared-memory parallelism.

3.1. Performance Impact of Nonsymmetry. We first examine the performance overhead incurred by the additional work for nonsymmetric compression. The nonsymmetric algorithm computes two approximations for each node β: a column ID of column off-diagonal block $K_{\tilde{\beta}\beta}$ and a row ID of row off-diagonal block $K_{\beta\tilde{\beta}}$. The symmetric algorithm only has to compute one of these and takes the transpose to get the other. We compare the results for the symmetric and nonsymmetric algorithms both applied to a symmetric problem. In these experiments, the operator is a kernel matrix $K \in \mathbb{R}^{n\times n}, n = 50000$, with entries defined by $K_{ij} = \mathcal{K}(x_i, x_j)$, where $\mathcal{K}$ is a Gaussian kernel function with bandwidth $h = 0.05$ and points $\{x_i\}$ are randomly drawn from a uniform distribution over the unit hypercube in four dimensions. Matrices such as this one appear in kernel methods in machine learning and their hierarchical rank structure has been demonstrated, for example in citeJLL:march2017far. The distribution of points and kernel bandwidth parameter are chosen so that the matrix is compressible, but not trivially so.

Table 3.1 shows performance results of the symmetric and nonsymmetric algorithms for various settings of the maximum approximation rank and leaf size.

When the approximation rank and leaf size are large, the cost of compression is dominated by the low-rank approximation of off-diagonal blocks, and the nonsymmetric compression takes roughly twice as long as the symmetric compression. As the approximation rank and leaf size are decreased, the low-rank approximation represents a smaller portion of the total compression time, and the penalty for nonsymmetric compression is much less than a factor of two. Setting these parameters too small results in poor performance due to increased costs of other steps. For the following experiments we use an approximation rank and leaf size of 256 unless otherwise noted.

While the nonsymmetric algorithm requires additional work for compression, it does not require any additional work for evaluation of the matrix-vector product. As expected, we observe no significant difference in evaluation times between the two algorithms.

3.2. Electromagnetic scattering on thin-slot geometry. Next, we consider a linear system generated by Eiger from an electromagnetic scattering problem on the thin-slot geometry shown in Figure 3.1 with $n = 15565$ unknowns. We compute approximate solutions to the system using GMRES with a compressed operator and with an uncompressed operator. The two algorithms were run until the relative residual was less than 1e$-$5. In the algorithm that uses the compressed operator, the convergence test uses the relative residual in the compressed operator $\|\tilde{K}\tilde{x} - b\|/\|b\|$. After the algorithm terminates, either due to satisfaction of the convergence condition or reaching the maximum number of iterations, the true relative residual is computed using the uncompressed operator $\|K\tilde{x} - b\|/\|b\|$.

Table 3.2 shows results for a right-hand side that was generated by applying the operator to a random vector drawn from a uniform distribution over $[-1, 1]^n$. For this problem, the

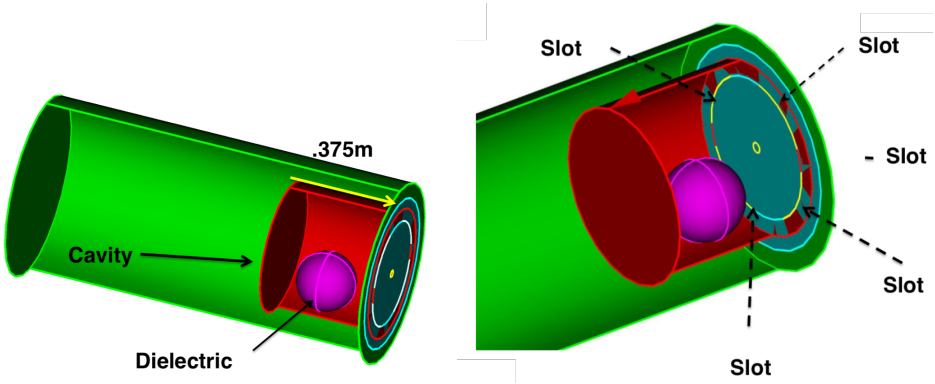


Fig. 3.1: Geometry of the thin-slot EM scattering problem [7].

Table 3.1: Timings of symmetric and nonsymmetric algorithms for compressed matrix-vector product of a (symmetric) Gaussian kernel matrix with a random vector ($n = 50000, \epsilon_c = 1\text{e}{-3}$, budget $= 0.05$). The compressed algorithms use the same value for both the maximum rank and maximum leaf-size parameters. The time for an uncompressed matrix-vector product is given for reference. Timings in seconds are reported separately for compression(Algorithm 6) and evaluation (Algorithm 7).

	Rank/leaf size	Compress time	Evaluate time	Rel. error
Symmetric	128	7.51	2.54	1.1e-3
Nonsymmetric	128	7.70	2.50	1.7e-3
Symmetric	256	1.42	0.62	1.5e-2
Nonsymmetric	256	1.62	0.58	1.4e-2
Symmetric	512	2.54	0.42	3.2e-2
Nonsymmetric	512	3.64	0.40	3.8e-2
Symmetric	1024	9.84	0.41	7.1e-2
Nonsymmetric	1024	19.80	0.44	6.0e-2
Uncompressed			2.15	

algorithm using the compressed operator took 0.35 seconds for compression and 0.72 seconds for 102 iterations of GMRES to reach a relative residual of 1e−5 in the compressed operator, corresponding to a relative residual of 3e−4 in the uncompressed operator. GMRES using the uncompressed operator completed after 101 iterations in 6.28 seconds, taking 8.8× longer per GMRES iteration and 5.9× longer in total runtime than the compressed algorithm. The two algorithms took nearly the same number of iterations to converge, but the GMRES iterations of the compressed algorithm were much faster.

Table 3.3 shows results for a given right-hand side of practical interest. In this case, both algorithms reached the limit of 200 iterations before convergence of the residual. For the compressed algorithm, the relative residual in the compressed operator reached 1e−2, but the true residual was 1.5. This discrepancy is caused by a large condition number ($\texttt{cond}(K) = 6.7\text{e}8$) compared to the relative error of the approximation $\left(\|K - \tilde{K}\|/\|K\| = 2.7\text{e}{-4}\right)$, and by the particular choice of b. The analysis in §2.5.1 shows that the discrepancy can be controlled

Table 3.2: GMRES applied to the linear system $Kx = b$, where K is the matrix from the thin-slot scattering problem and b is set to be product of K applied to a random vector drawn from a uniform distribution over $[0, 1]^n$. The table reports compression time in seconds, GMRES time in seconds (excluding compression), the number of GMRES iterations, and relative residual in the uncompressed operator.

	Compress time	GMRES time	# iter	True residual
Compressed	0.35	0.72	102	3e-4
Uncompressed	0	6.28	101	1e-5

Table 3.3: GMRES applied to the linear system $Kx = b$, where K is the matrix from the thin-slot scattering problem and b is a given right-hand side of practical interest. The table reports compression time in seconds, GMRES time in seconds (excluding compression), the number of GMRES iterations, and relative residual in the uncompressed operator.

	Compress time	GMRES time	# iter	True residual
Compressed	0.36	2.1	200	1.5
Uncompressed	0	12.3	200	6e-3

by ensuring $\epsilon_c \texttt{cond}(K)$ is much less than 1, which is not satisfied.

A possible remedy would be to compress the matrix to higher accuracy, but attempts to tighten the error tolerance parameter failed to produce a more accurate compression. The cause of this failure is likely to be poor quality of the samples used in the sampled ID, which were selected at random due to the absence of geometric information. Geometric information can be easily incorporated into the compression algorithm, but it must first be extracted during construction of the discretized problem, and such information was not available at the time of writing. To demonstrate this, we run the algorithm with sampling and without sampling over a range of values for the error tolerance parameter as shown in Table 3.4. The error decreases along with the error tolerance for the unsampled algorithm but not for the sampled algorithm. The unsampled algorithm gives better convergence for this problem, but it is not practical due to its expensive compression phase, which requires $\mathcal{O}(n^2)$ operations. For example, with $\epsilon_c/\|K\| = 1\text{e}{-9} \approx \text{cond}(K)$, the compression time without sampling is 70.5 seconds, which is enough time to run thousands of GMRES iterations using the uncompressed operator. Though we cannot know the extent to which geometric information would improve sampling, it has been observed that the sampling scheme we use generally works well on a range of practical problems, and uniform random sampling generally performs poorly [9].

3.3. Helmholtz kernel. We consider another problem for which geometric information is available. In these experiments, we use a kernel matrix defined by the three-dimensional Helmholtz kernel

$$\mathcal{K}(x_i, x_j) = \frac{e^{-ik\|x_i - x_j\|_2}}{\|x_i - x_j\|_2}, \tag{3.1}$$

where the points $\{x_i\}_{i=1}^n$ are drawn from a uniform distribution on $[0, 1]^3$. A large wave number k in the kernel function produces a highly oscillatory kernel function, resulting in a matrix that is difficult to compress. We solve the regularized linear least squares problem $(DK + \lambda I)x = b$, where D is a diagonal matrix with diagonal entries drawn randomly from

Table 3.4: Comparison of compression accuracy using a sampled ID versus an unsampled ID for the scattering problem. The first column reports the relative error tolerance, and the second and third columns report the relative error of the compressed operator constructed using either sampled IDs or unsampled IDs.

Rel. error tolerance	Sampled ID rel. error	Unsampled ID rel. error
1e-4	3e-4	3e-4
1e-5	3e-4	1e-4
1e-6	1e-3	3e-5
1e-7	1e-3	6e-7
1e-8	2e-3	2e-8
1e-9	4e-3	9e-10
1e-10	4e-3	7e-11

Table 3.5: Matrix-vector product $(DK + \lambda I)x$, where K is the Helmholtz kernel matrix and x is drawn from a uniform distribution over $[0, 1]^n$. The first column reports the relative error tolerance, the second and third columns report timings in seconds for compression and evaluation, and the fourth column reports the relative error of the compressed operator.

Rel. error tolerance	Compress time	Evaluate time	Rel. error
1e-1	1.51	0.024	4e-1
1e-2	1.53	0.025	6e-2
1e-3	1.76	0.028	2e-2
1e-4	1.84	0.031	1e-3
1e-5	1.92	0.048	9e-5
1e-6	2.05	0.073	6e-6
1e-7	2.23	0.116	1e-6
1e-8	2.33	0.253	2e-7
1e-9	2.69	0.369	4e-8
1e-10	3.10	0.577	1e-8
Uncompressed		0.252	

a uniform distribution on $[0, 1]$ and $\lambda \in \mathbb{R}$ is a regularization term. The purpose of applying D is to make the matrix $DK + \lambda I$ nonsymmetric, and the regularization term is chosen to control the rate of convergence of GMRES. In these experiments, $n = 32768, \lambda = 300$.

Table 3.5 shows performance and accuracy of the matrix-vector product for a range of error tolerances. As the error tolerance is decreased, the approximation error of the compressed operator decreases accordingly, and costs increase for both compression and evaluation. Nearly every case demonstrates speedup over the uncompressed matrix-vector product.

Table 3.6 shows performance and accuracy of GMRES applied to the linear system $(DK + \lambda I)x = b$, where b is defined to be the product of $DK + \lambda I$ applied to a random vector drawn from a uniform distribution over $[0, 1]^n$. The problem is solved using an compressed operator for a range of error tolerances and with the uncompressed operator. We report the final relative residual both in the compressed operator and in the uncompressed operator. In cases with larger error tolerance, the residual in the uncompressed operator remains large even after convergence in the compressed operator due to error in the compression. For more accurate compressed operators, this discrepancy diminishes, and the two residuals reach

Table 3.6: GMRES applied to the linear system $(DK + \lambda I)x = b$, where K is the Helmholtz kernel matrix and b is set to be product of $DK + \lambda I$ applied to a random vector drawn from a uniform distribution over $[0, 1]^n$. The table reports the relative error tolerance, compression time in seconds, GMRES time in seconds (excluding compression), relative residual in the compressed operator, and the true residual (the relative residual in the uncompressed operator).

Rel. err. tol.	Compress time	GMRES time	Comp. resid.	True resid.
1e-1	1.5	3.0	6e-4	2e-1
1e-2	1.5	3.3	3e-5	4e-2
1e-3	1.8	3.8	2e-5	2e-2
1e-4	1.8	4.7	2e-5	3e-3
1e-5	1.9	5.7	2e-5	4e-4
1e-6	2.0	11.5	2e-5	7e-5
1e-7	2.2	16.8	1e-5	2e-5
1e-8	2.3	30.0	2e-5	2e-5
1e-9	2.7	36.8	2e-5	2e-5
1e-10	3.1	45.4	2e-5	2e-5
Uncompressed		25.6	2e-5	2e-5

agreement.

4. Conclusions and Future Work. We present an algorithm for compressing and applying rank-structured nonsymmetric matrices such as those arising from applying the method of moments to the combined field integral equation. We integrate the compression scheme with an iterative solver and apply it to an electromagnetic scattering problem, demonstrating significant speedup over an iterative solver that uses direct matrix-vector products. We also explore cases for which the algorithm performs poorly, for example, in the absence of geometric information and for a system with a right-hand side that is difficult to solve. Directions for further development include tighter integration between the GOFMM and Eiger (or the new Gemma) codes so that the problem can be solved in a matrix-free manner, applying a preconditioner or scaling rows and columns to solve the problem faster and more robustly, the use of inexact Krylov methods for further speedup, and using the distributed-memory algorithm [19] to solve much larger problems.

REFERENCES

[1] F. P. Andriulli, K. Cools, I. Bogaert, and E. Michielssen, *On a well-conditioned electric field integral operator for multiply connected geometries*, IEEE transactions on antennas and propagation, 61 (2013), pp. 2077–2087.

[2] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, *Amesos2 and belos: Direct and iterative solvers for large sparse linear systems*, Scientific Programming, 20 (2012), pp. 241–255.

[3] H. Cheng, W. Y. Crutchfield, Z. Gimbutas, L. F. Greengard, J. F. Ethridge, J. Huang, V. Rokhlin, N. Yarvin, and J. Zhao, *A wideband fast multipole method for the helmholtz equation in three dimensions*, Journal of Computational Physics, 216 (2006), pp. 300–325.

[4] H. Guo, Y. Liu, J. Hu, and E. Michielssen, *A butterfly-based direct integral-equation solver using hierarchical lu factorization for analyzing scattering from electrically large conducting objects*, IEEE Transactions on Antennas and Propagation, 65 (2017), pp. 4742–4750.

[5] M. A. Heroux and J. M. Willenbring, *A new overview of the trilinos project*, Scientific Programming, 20 (2012), pp. 83–88.

[6] W. A. Johnson et al., *Eiger™: An open-source frequency-domain electromagnetics code*, in Antennas and Propagation Society International Symposium, 2007 IEEE, IEEE, 2007, pp. 3328–3331.

[7] J. D. Kotulski, *Computational electromagnetics at sandia national laboratories - current code capability.*, (2015).

[8] W. L. Langston, J. Kotulski, R. Coats, R. Jorgenson, S. A. Blake, S. Campione, A. Pung, and B. Zinser, *Massively parallel frequency domain electromagnetic simulation codes*, in Applied Computational Electromagnetics Society Symposium (ACES), 2018 International, IEEE, 2018, pp. 1–2.

[9] W. B. March and G. Biros, *Far-field compression for fast kernel summation methods in high dimensions*, Applied and Computational Harmonic Analysis, 43 (2017), pp. 39–75.

[10] W. B. March, B. Xiao, S. Tharakan, D. Y. Chenhan, and G. Biros, *A kernel-independent fmm in general dimensions*, in High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for, IEEE, 2015, pp. 1–12.

[11] W. B. March, B. Xiao, C. D. Yu, and G. Biros, *Askit: an efficient, parallel library for high-dimensional kernel summations*, SIAM Journal on Scientific Computing, 38 (2016), pp. S720–S749.

[12] J. R. Mautz and R. F. Harrington, *H-field, e-field, and combined field solutions for bodies of revolution*, tech. rep., SYRACUSE UNIV NY DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 1977.

[13] S. Rao, D. Wilton, and A. Glisson, *Electromagnetic scattering by surfaces of arbitrary shape*, IEEE Transactions on antennas and propagation, 30 (1982), pp. 409–418.

[14] Y. Saad and M. H. Schultz, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on scientific and statistical computing, 7 (1986), pp. 856–869.

[15] X. Q. Sheng, J.-M. Jin, J. Song, W. C. Chew, and C.-C. Lu, *Solution of combined-field integral equation using multilevel fast multipole algorithm for scattering by homogeneous bodies*, IEEE transactions on antennas and propagation, 46 (1998), pp. 1718–1726.

[16] J. Song and W. C. Chew, *Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering*, Microwave and Optical Technology Letters, 10 (1995), pp. 14–19.

[17] J. Song, C.-C. Lu, and W. C. Chew, *Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects*, IEEE Transactions on Antennas and Propagation, 45 (1997), pp. 1488–1493.

[18] C. D. Yu, J. Levitt, S. Reiz, and G. Biros, *Geometry-oblivious fmm for compressing dense spd matrices*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 53.

[19] C. D. Yu, S. Reiz, and G. Biros, *Distributed-memory hierarchical compression of dense spd matrices*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, IEEE Press, 2018, p. 15.

ON DIFFERENTIABLE LINEARITY AND LOCAL BOUNDS PRESERVING STABILIZATION METHODS FOR FIRST ORDER CONSERVATION LAW SYSTEMS

JESUS BONILLA*, SIBUSISO MABUZA†, JOHN N. SHADID‡, AND SANTIAGO BADIA§

Abstract. In this work, a new stabilized continuous finite element scheme for first order conservation laws is presented. The proposed method is shown to be local bounds preserving for first order conservation law systems. In addition, it is also shown to satisfy the discrete maximum principle (DMP) for steady scalar convection equations. This scheme is based on the combination of an artificial diffusion operator and a differentiable, linearity preserving shock detector in what is usually termed algebraic flux correction (AFC). This work extends some ideas on differentiable nonlinear stabilization, which have been recently developed for scalar transport problems. The behavior of the numerical scheme is analyzed qualitatively and quantitatively for scalar convection and a very preliminary solution for a steady Euler equation problem.

1. Introduction. Hyperbolic conservation laws such as Euler equations are expected to satisfy constraints such as the positivity of the density and the internal energy. Discretizations that do not satisfy these constraints may yield non-physical solutions. Constructing robust and flexible *stabilized* schemes that satisfy the above conditions continues to be a challenge. A number of stabilized schemes have been considered in the context of finite volume [14], discontinuous Galerkin (DG) [1] and continuous finite element (FE) [10] methods. Finite volume methods become complex when trying to achieve high order convergence with large element stencils involved. However, this problem can be addressed using arbitrarily high order DG methods. Recent progress has been made in stabilized FE schemes by making use of FCT algorithms [10, 15–17],VMS [5], SUPG [8] and entropy viscosity [7] among others. In this context FCT methods have shown promise. In [8] these methods were shown to perform better than residual based schemes such as SUPG, constraining the problem in a way that preserves positivity for scalar transport problems.

In this work, we are interested in designing a stabilized scheme that will be robust for steady and transient shock hydrodynamics problems. In particular, we focus on fully implicit problems that will involve nonlinear solution strategies. The fully discrete problem may be solved using Newton's method. We therefore consider that the stabilization part of the scheme needs to be smooth enough for convergence of the nonlinear solver to be achieved easily. One solution for this is differentiable AFC stabilization. Recently, differentiable AFC stabilization based on limited artificial diffusion has been successfully applied to scalar problems discretized with continuous Galerkin and DG methods [2, 3]. The results showed improved convergence of the nonlinear solver. In this work, we extend the above ideas to the linearity preserving AFC stabilization presented in [9] and the new scheme is applied to steady convection and first order conservation law equations.

This paper is structured as follows: In section 2 we present the CG discretization for scalar convection and Euler equations. Section 3 is devoted to the differentiable AFC stabilization term. Then, in section 4 we present some numerical results. Finally, we present the conclusion and summary in section 5.

2. Preliminaries.

* Universitat Politècnica de Catalunya & Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), jbonilla@cimne.upc.edu

†Center for Computing Research, Sandia National Laboratories, smabuza@sandia.gov

‡Center for Computing Research, Sandia National Laboratories, & Department of Mathematics and Statistics, University of New Mexico, jnshadi@sandia.gov

§Universitat Politècnica de Catalunya & Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), sbadia@cimne.upc.edu

2.1. Continuous problem. Let Ω be an open, bounded and connected subset of $\mathbb{R}^d$, where d is the number of spatial dimensions. Let $\partial\Omega$ be the boundary which is Lipschitz continuous. Consider a first order hyperbolic system given by

$$\begin{cases} \partial_t \boldsymbol{u} - \boldsymbol{\nabla}\cdot\boldsymbol{f}(\boldsymbol{u}) = \boldsymbol{g}, & \text{in } \Omega\times(0,T], \\ u^\beta(x,t) = \bar{u}^\beta(x,t), & \text{on } \Gamma^\beta_{\text{in}}\times(0,T],\ \beta = 1,...,m, \\ \boldsymbol{u}(x,0) = \boldsymbol{u}_0(x), & x\in\Omega, \end{cases} \tag{2.1}$$

where $\boldsymbol{u} = \{u^\beta\}_{\beta=1}^m$ are $m \geq 1$ conserved variables, $\boldsymbol{f}$ is the physical flux. Note that $\boldsymbol{f}:\mathbb{R}^m\to\mathbb{R}^{m\times d}$, $\boldsymbol{f} = \{\boldsymbol{f}_i\}_{i=1}^d$ and $\boldsymbol{f}_i:\mathbb{R}^m\to\mathbb{R}^m$. $\bar{u}^\beta(x,t)$ are the boundary values for the βth-component of $\boldsymbol{u}$, $\boldsymbol{u}_0(x)$ are the initial conditions, and $\boldsymbol{g}(x,t)$ is a function defining the body forces. The inflow boundary for component β is defined as $\Gamma^\beta_{\text{in}} \doteq \{\boldsymbol{x}\in\partial\Omega\ : \lambda^\beta(\boldsymbol{f}'(\boldsymbol{u})\cdot\boldsymbol{n}_{\partial\Omega}) \leq 0\}$, where $\boldsymbol{n}_{\partial\Omega}$ is the unit outward normal to the boundary, and λ^β is the βth-eigenvalue of the flux Jacobian. We define the outflow boundary as $\Gamma^\beta_{\text{out}} \doteq \partial\Omega\backslash\Gamma^\beta_{\text{in}}$. We will also consider the steady counterpart of (2.1), that is obtained by dropping the time derivative term and the initial conditions. Note that if $m = 1$, and $\boldsymbol{f}(u) \doteq \mathbf{v}u$ with $\mathbf{v}$ a divergence-free convection field, we recover the well known scalar convection problem. On the other hand, Euler equations are recovered when we define $\boldsymbol{u} \doteq [\rho, \boldsymbol{m}, \rho E]^T$, $\boldsymbol{f}_i \doteq m_i[1, \mathbf{v}, E] + p[1, \delta_{1i}, \ldots, \delta_{di}, v_i]$, and $\boldsymbol{g} \doteq [0, \boldsymbol{b}, \boldsymbol{b}\cdot\mathbf{v} + r]$, where ρ is the density, E is the total energy, p is the pressure, $\boldsymbol{m} = \{m_1,\ldots,m_d\}$, where $m_i = \rho v_i$, is the momentum, $\mathbf{v} = \{v_1,\ldots,v_d\}$ is the velocity, $\boldsymbol{b} = \{b_1,\ldots,b_d\}$ are the body forces, r is the heat supply per unit mass, and δ_{ij} is the well known Kronecker delta.

2.2. Discretization. Let $\mathcal{T}_h$ be a conforming partition of Ω into elements, K. The set of interpolation nodes, i, at coordinates, $\boldsymbol{x}_i$, of mesh $\mathcal{T}_h$ is represented by $\mathcal{N}_h$, whereas the set of nodes belonging to a particular element is defined as $\mathcal{N}_h(K) \doteq \{i\in\mathcal{N}_h : \boldsymbol{x}_i\in K\}$. Moreover, Ω_i is the macroelement composed by the union of elements that contain node i, that is, $\Omega_i \doteq \bigcup_{K\in\mathcal{T}_h,\ \boldsymbol{x}_i\in K} K$. To simplify the discussion below, we abuse notation and use i for both the node and its associated index.

We define the finite element space as $\boldsymbol{V}_h^p \doteq \left\{\boldsymbol{v}_h\in(\mathcal{C}^0)^m : \boldsymbol{v}_h|_K\in(\mathcal{P}_p^d)^m, \forall K\in\mathcal{T}_h\right\}$ for simplex partitions of Ω or $\boldsymbol{V}_h^p \doteq \left\{\boldsymbol{v}_h\in(\mathcal{C}^0)^m : \boldsymbol{v}_h|_K\in(Q_p^d)^m, \forall K\in\mathcal{T}_h\right\}$ for d-cube partitions, where d is the number of dimensions and m the number of components of $\boldsymbol{u}$. Further, we define the space $\boldsymbol{V}_{h0}^p \doteq \{\boldsymbol{v}_h\in\boldsymbol{V}_h^p \mid \boldsymbol{v}_h(\boldsymbol{x}) = 0\ \forall\boldsymbol{x}\in\Gamma_{\text{in}}\}$. We restrict the present work to first order finite elements, i.e. $p = 1$. Henceforth we drop the exponent p form the finite element spaces. Also, denote by V_h the space $\boldsymbol{V}_h$ when $m\equiv 1$. The functions $\boldsymbol{v}_h\in\boldsymbol{V}_h$ can be constructed as a linear combination of the basis $\{\varphi_i\}_{i\in\mathcal{N}_h}$ and nodal values $\boldsymbol{v}_i$, where φ_i is the shape function associated to the node i. Hence, $\boldsymbol{v}_h = \sum_{i\in\mathcal{N}_h}\varphi_i\boldsymbol{v}_i$.

The $L_2(\omega)$ scalar product is denoted by $(\cdot,\cdot)_\omega$ for $\omega\subset\Omega$, however we omit the subscript for $\omega\equiv\Omega$. The $L_2(\Omega)$ norm is denoted by $\|\cdot\|$. Furthermore, we denote by $\mathbb{1}$ the function that is equal to 1 in Ω; $\mathbb{1}(\boldsymbol{x}) = 1\ \forall\boldsymbol{x}\in\Omega$.

The semi-discrete form of problem (2.1) reads:
Find $\boldsymbol{u}_h\in\boldsymbol{V}_h$ such that $u_h^\beta = \bar{u}_h^\beta$ on Γ^β_{in}, $\boldsymbol{u}_h = \boldsymbol{u}_{0h}$ at $t = 0$, and

$$(\partial_t\boldsymbol{u}_h, \boldsymbol{v}_h) + (\boldsymbol{u}_h, \boldsymbol{f}'(\boldsymbol{u}_h) : \boldsymbol{\nabla}\boldsymbol{v}_h) - (\boldsymbol{u}_h, \boldsymbol{n}_{\Gamma_{\text{out}}}\cdot\boldsymbol{f}'(\boldsymbol{u}_h)\boldsymbol{v}_h)_{\Gamma_{\text{out}}} = (\boldsymbol{g}, \boldsymbol{v}_h),\ \text{for all } \boldsymbol{v}_h\in\boldsymbol{V}_{h0}, \tag{2.2}$$

where $\bar{u}_h^\beta$ and $\boldsymbol{u}_{0h}$ are finite element approximations of $\bar{u}^\beta$ and $\boldsymbol{u}_0$ such that the DMP is satisfied.

For the sake of brevity, we only consider backward Euler time discretization (BE). Higher order SSP RK time integration can also be used [6]. In that case, a prerequisite for satisfying

monotonicity properties is that CFL-like conditions be fulfilled [11, 13]. Consider a partition of the time domain $(0, T]$ into n^{ts} sub-intervals of length $(t^n, t^{n+1}]$. Then, for every time step, n, we solve

$$\mathsf{M}\delta_t \mathsf{U}^{n+1} + \mathsf{K}\mathsf{U}^{n+1} = \mathsf{G} \tag{2.3}$$

where U_j^{n+1} is the vector of nodal values at time $t = t^{n+1}$, $\delta_t(\cdot) \doteq \Delta t_{n+1}^{-1}((\cdot)^{n+1} - (\cdot)^n)$, and $\Delta t_{n+1} \doteq |t^{n+1} - t^n|$. The $m \times m$-matrices relating nodes i, j are given by $\mathsf{M}_{ij} \doteq (\varphi_j, \varphi_i) I_{m\times m}$, $\mathsf{K}_{ij}^{\beta\gamma} \doteq (\varphi_j \delta_{\beta\xi}, \boldsymbol{f}'(\boldsymbol{u}_h)_k^{\xi\eta} \cdot \partial_k \varphi_i \delta_{\eta\gamma}) - (\varphi_j \delta_{\beta\xi}, n_k \cdot \boldsymbol{f}'(\boldsymbol{u}_h)_k^{\xi\eta} \varphi_i \delta_{\eta\gamma})_{\Gamma_{\text{out}}}$, where $\beta, \gamma, \xi, \eta = 1, \ldots, m$ are the component indices, and $\mathsf{G}_i \doteq (\boldsymbol{g}, \varphi_i)$ for $i, j \in \mathcal{N}_h$.

2.3. Monotonicity properties. In this section, we define all the properties that we demand our FE scheme to fulfill. Let us firstly introduce them for the case of a scalar problem (i.e. $m = 1$).

Definition 2.1 (Local Discrete Extremum). *The function $v_h \in V_h$ has a local discrete minimum (resp. maximum) on $i \in \mathcal{N}_h$ if $u_i \leq u_j$ (resp. $u_i \geq u_j$) $\forall j \in \mathcal{N}_h(\Omega_i)$.*

Definition 2.2 (Local DMP). *A solution $u_h \in V_h$ satisfies the local discrete maximum principle if for every $i \in \mathcal{N}_h$*

$$\min_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} u_j \leq u_i \leq \max_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} u_j. \tag{2.4}$$

Definition 2.3 (LED). *A scheme is local extremum diminishing if, for every u_i that is a local discrete maximum (resp. minimum),*

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} \leq 0, \qquad \left(\textit{resp. } \frac{\mathrm{d}u_i}{\mathrm{d}t} \geq 0\right), \tag{2.5}$$

is satisfied.

In the case of vector problems, Defs. 2.1–2.2 are simply applied for every component. E.g. $\boldsymbol{u}_h \in \boldsymbol{V}_h$ will have an extremum at component β if Def. 2.1 is satisfied for some u_i^β. However, Kuzmin and co-workers [10] propose another definition of LED in the case of vector valued problems. This definition is extended from the LED theory for scalar problems. It is known that a scheme with a positive diagonal mass matrix and a stiffness matrix with the M-matrix property (e.g. $K_{ij} \leq 0$ for $j \neq i$, and $K_{ii} \geq \sum_{j\neq i} -K_{ij}$) is LED (see [13]). In an attempt to enforce Def. 2.3 for every characteristic variable of the vector problem, the LED property is redefined as:

Definition 2.4 (LED adapted from [10]). *A problem is said to be LED if for every $i, j \in \mathcal{N}_h$, the off-diagonal blocks, i.e. $j \neq i$, of the stiffness matrix are negative semi-definite, and the mass matrix is diagonal.*

It is worth noting that the above definition of LED is less restrictive for vector problems. As opposed to Def. 2.3, the definition above does not imply that all extrema are diminishing in terms of the unknown, $\boldsymbol{u}_h$. If Def. 2.4 is satisfied, then the characteristic variables of the linearized problem satisfy Def. 2.3. Hence, only local bounds are preserved. See [10] for details.

Finally, let us define linearity-preservation, which is a property required to achieve optimal convergence rates in smooth regions see [12].

Definition 2.5 (Linearity-preservation). *A stabilization term, say $\mathsf{B}_{ij}(\boldsymbol{u}_h)$, is said to be linearity-preserving if for all $\beta \in C$, $u_h^\beta \in \mathcal{P}_1^d(\Omega)$, then $\mathsf{B}_{ij}(\boldsymbol{u}_h) = 0$, where C is the set of components used to detect inadmissible values of $\boldsymbol{u}_h$.*

3. Nonlinear stabilization. In this section we define a stabilization term, $\mathsf{B}_{ij}(\boldsymbol{u}_h)$, to be added to the discrete problem (2.2) such that it satisfies Def. 2.2 for steady scalar problems, Def. 2.3 in the case of transient scalar problems, and Def. 2.4 for a hyperbolic system of equations. Furthermore, we enforce that it;

1. has compact support: $\mathsf{B}_{ij} = 0$ if $j \notin \mathcal{N}_h(\Omega_i)$,
2. is symmetric: $\mathsf{B}_{ij}(\boldsymbol{u}_h) = \mathsf{B}_{ji}(\boldsymbol{u}_h)$,
3. is conservative: $\sum_{j\neq i} \mathsf{B}_{ij}(\boldsymbol{u}_h) = -\mathsf{B}_{ii}(\boldsymbol{u}_h)$,
4. is linearity-preserving (see Def. 2.5),
5. is twice-differentiable $\frac{\partial^2 \mathsf{B}_{ij}(\boldsymbol{u}_h)}{\partial \boldsymbol{u}_h^2} \in \mathcal{C}^0$.

In order to achieve these requirements we present a stabilization term constructed in the following way: First, we define a shock detector based on the design in [9], and use the ideas in [2] to improve its nonlinear convergence. In this, we regularize non-differentiable functions in stabilization term to obtain a twice differentiable version.

Recall the set of regularized functions used in [2] to satisfy requirement (5) in the previous list. Absolute values are regularized as follows

$$|x|_{1,\varepsilon_h} = \sqrt{x^2 + \varepsilon_h}, \qquad |x|_{2,\varepsilon_h} = \frac{x^2}{\sqrt{x^2 + \varepsilon_h}}. \tag{3.1}$$

Note that $|x|_{2,\varepsilon_h} \leq |x| \leq |x|_{1,\varepsilon_h}$. Next, we define a smooth maximum function, $\max_{\sigma_h}(\cdot)$, as

$$\max\nolimits_{\sigma_h}\{x, y\} \doteq \frac{|x-y|_{1,\sigma_h}}{2} + \frac{x+y}{2}, \tag{3.2}$$

We also use the following function to limit the value of any given quantity to one

$$Z(x) \doteq \begin{cases} 2x^4 - 5x^3 + 3x^2 + x, & x < 1, \\ 1, & x \geq 1. \end{cases} \tag{3.3}$$

Finally, we define $H_0^{\tau_h}(x)$ as a regularized version of the well known Heaviside function,

$$H_0^{\tau_h}(x) \doteq \begin{cases} 1, & \text{if} \quad x \geq \tau_h, \\ \frac{6}{\tau_h^5}(x - \frac{1}{2}\tau_h)^5 - \frac{5}{\tau_h^3}(x - \frac{1}{2}\tau_h)^3 + \frac{15}{8\tau_h}(x - \frac{1}{2}\tau_h), & \text{if} \quad 0 < x < \tau_h, \\ 0, & \text{if} \quad x \leq 0. \end{cases} \tag{3.4}$$

The proposed stabilization term is given by

$$B_h(\boldsymbol{w}_h; \boldsymbol{u}_h, \boldsymbol{v}_h) \doteq \sum_{K^e \in \mathcal{T}_h} \sum_{i,j \in \mathcal{N}_h(K^e)} \nu_{ij}^e(\boldsymbol{w}_h) \ell(i,j) \boldsymbol{v}_i \cdot I_{m\times m} \boldsymbol{u}_j, \tag{3.5}$$

for any $\boldsymbol{u}_h \in \boldsymbol{V}_h$ and $\boldsymbol{v}_h \in \boldsymbol{V}_{h0}$. Here, $\ell(i,j) \doteq 2\delta_{ij} - 1$ is a graph Laplacian operator as defined in [2], and $\nu_{ij}^e(\boldsymbol{w}_h)$ is the element-wise artificial diffusion defined as

$$\begin{aligned} \nu_{ij}^e(\boldsymbol{w}_h) &\doteq \alpha^e(\boldsymbol{w}_h) \max\nolimits_{\sigma_h}\{\lambda_{ij}^{\max}, \lambda_{ji}^{\max}\}, \quad \text{for } j \in \mathcal{N}_h(\Omega_i)\backslash\{i\}, \\ \nu_{ii}^e(\boldsymbol{w}_h) &\doteq \sum_{j \in \mathcal{N}_h(\Omega_i)\backslash\{i\}} \nu_{ij}^e(\boldsymbol{w}_h), \end{aligned} \tag{3.6}$$

where, $\lambda_{ij}^{\max}$ is the spectral radius of the elemental matrix relating nodes $i, j \in \mathcal{N}_h$. This artificial diffusion is based on Rusanov scalar diffusion [10]. We denote by $\alpha^e(\boldsymbol{w}_h)$ the shock detector used for computing the artificial diffusion parameter. The idea behind the definition of this detector is ensuring that Def. 2.2–2.5 are satisfied using minimal amount of artificial

diffusion. This $\alpha^e(\boldsymbol{w}_h)$ must be a positive real number which takes value 1 when $\boldsymbol{u}_h(\boldsymbol{x}_i)$ is an inadmissible value of $\boldsymbol{u}_h$, and smaller than 1 otherwise. To this end, we define

$$\alpha^e \doteq \min_\sigma\{\Phi_i^\beta\}_{i\in\mathcal{N}_h(K^e)}^{\beta\in C}, \tag{3.7}$$

where C is the set of components that are used to detect inadmissible values of $\boldsymbol{u}_h$, e.g. density and total energy in the case of Euler equations. For simplicity, we restrict ourselves to the components of $\boldsymbol{u}_h$, however derived quantities such as the pressure can be used. The nodal detector, Φ_i^β, is defined as

$$\Phi_i^\beta(\boldsymbol{u}_h) \doteq \left[Z\left(\frac{\left|\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta (u_i^\beta - u_j^\beta)\right|_{1,\varepsilon_h} + \gamma_h}{\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta \left|u_i^\beta - u_j^\beta\right|_{2,\varepsilon_h} + \gamma_h} \right) \right]^q, \tag{3.8}$$

where q is a parameter to modulate the amount of diffusion added –the higher is q the less diffusion is added, see Sect. 4 or [2,3] for experimental analysis of the effect of q–, ω_{ij}^β is a correction factor used to satisfy linearity-preservation (see Def. 2.5), and it reads

$$\omega_{ij}^\beta \doteq \frac{H_0^{\tau_h}(\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j))}{S_+^\beta + H_0^{\tau_h}(-S_+^\beta+\tau_h)} + \frac{H_0^{\tau_h}(-\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j))}{S_-^\beta + H_0^{\tau_h}(-S_-^\beta+\tau_h)} \tag{3.9}$$

$$+ H_0^{\tau_h}(\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j)+\tau_h) H_0^{\tau_h}(-\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j)+\tau_h), \tag{3.10}$$

where

$$S_+^\beta \doteq \sum_{j\in\mathcal{N}_h(\Omega_i)} H_0^{\tau_h}(\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j))\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j), \tag{3.11}$$

$$S_-^\beta \doteq \sum_{j\in\mathcal{N}_h(\Omega_i)} -H_0^{\tau_h}(-\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j))\mathbf{g}_i^\beta\cdot(\boldsymbol{x}_i-\boldsymbol{x}_j), \tag{3.12}$$

and $\mathbf{g}_i^\beta$ is the lumped mass L_2 projection of the gradient of u_h^β, defined by $\mathbf{g}_i^\beta \doteq \frac{1}{(\mathbb{1},\varphi_i)}(\boldsymbol{\nabla} u_h^\beta,\varphi_i)$.

In addition to limiting the artificial diffusion, we do a selective lumping of the mass matrix associated with the time derivative. This selective lumping is driven by the same shock detector used in the artificial diffusion operator, $B_h(\boldsymbol{u}_h;\boldsymbol{u}_h,\boldsymbol{v}_h)$. This is achieved by replacing M in (2.3) by $\tilde{\mathsf{M}} = \{\tilde{\mathsf{M}}_{ij}\}_{ij}$ defined by $\tilde{\mathsf{M}}_{ij}(\boldsymbol{u}_h) \doteq \sum_{K^e\in\mathcal{T}_h}\sum_{i,j\in\mathcal{N}_h(K^e)}(1-\alpha^e)(\varphi_j,\varphi_i)I_{m\times m} + \alpha^e(\mathbb{1},\varphi_i)I_{m\times m}$.

The final stabilized problem in matrix form reads:

Find $\boldsymbol{u}_h \in \boldsymbol{V}_h$ such that $\boldsymbol{u}_h = \overline{\boldsymbol{u}}_h$ on $\partial\Omega$, $\boldsymbol{u}_h = \boldsymbol{u}_{0h}$ at $t=0$, and

$$\tilde{\mathsf{M}}\delta_t\mathsf{U}^{n+1} + \tilde{\mathsf{K}}_{ij}(\boldsymbol{u}_h^{n+1})\mathsf{U}^{n+1} = \mathsf{G} \tag{3.13}$$

for $n=1,...,n^{ts}$, where $\tilde{\mathsf{K}}_{ij}(\boldsymbol{u}_h) \doteq \mathsf{K}_{ij} + \mathsf{B}_{ij}(\boldsymbol{u}_h)$, and $\mathsf{B}_{ij}(\boldsymbol{u}_h) = B_h(\boldsymbol{u}_h;\varphi_j,\varphi_i)$, for $i,j\in\mathcal{N}_h$.

We will now show that the stabilization term defined in (3.5) satisfies all properties defined in Sect. 2.3.

LEMMA 3.1. *The shock detector defined in (3.7) is $\alpha^e = 1$ if component $\beta\in C$ of $\boldsymbol{u}_h$ has a local discrete extremum at $\boldsymbol{x}_i$ for $i\in\mathcal{N}_h(K^e)$, and otherwise is valued $0\le\alpha^e<1$ if $\varepsilon_h\to 0$ and $\gamma_h\to 0$.*

Proof. Let us start proving that the shock detector is equal to 1 if u_h^β has an extremum at $\boldsymbol{x}_i$ for $i\in\mathcal{N}_h(K^e)$. It is obvious that if $\Phi_i^\beta = 1$ for $\boldsymbol{u}_h$ with an extremum at $\boldsymbol{x}_i$ then

$\alpha^e = 1$, thus it is enough to prove the former. To this end, it is useful to prove that $\omega_{ij}^\beta > 0$ for all $i, j \in \mathcal{N}_h$. By definition $0 < H_0^{\tau_h}(\xi) \leq 1$ for $\xi > 0$, then it is easy to see that $S_+^\beta \geq 0$ and $S_-^\beta \geq 0$. Further, it can be seen that all terms in the definition of ω_{ij}^β are nonnegative. Moreover, it is easy to check that for any combination of ij at least one of its terms is not null, hence $\omega_{ij}^\beta > 0$.

Without loss of generality, let us assume that u_h^β has a local discrete maximum at $\boldsymbol{x}_i$, i.e. $u_i^\beta \geq u_j^\beta$ for $j \in \mathcal{N}_h(\Omega_i)$, then

$$\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta \left|u_i^\beta - u_j^\beta\right|_{2,\varepsilon_h} \leq \sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta |u_i^\beta - u_j^\beta| = \sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta (u_i^\beta - u_j^\beta) \tag{3.14}$$

$$= \left|\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta (u_i^\beta - u_j^\beta)\right| \leq \left|\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta (u_i^\beta - u_j^\beta)\right|_{1,\varepsilon_h}. \tag{3.15}$$

Therefore, the quotient in (3.8) is larger or equal to 1. This quotient is the argument of function $Z(x)$, which is bounded above by 1 (see (3.3)). Thus, $\Phi_i = [Z(x)]^q = 1$ if u_i^β is a maximum. Proceeding analogously for a minimum we conclude that $\Phi_i = 1$ if u_i^β is a local discrete extremum. Otherwise, if u_h^β does not have an extremum at $\boldsymbol{x}_i$ for $i \in K^e$ and $\varepsilon_h \to 0$,

$$\lim_{\varepsilon_h\to 0} \sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta \left|u_i^\beta - u_j^\beta\right|_{2,\varepsilon_h} = \sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta |u_i^\beta - u_j^\beta| \tag{3.16}$$

$$> \left|\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta (u_i^\beta - u_j^\beta)\right| = \lim_{\varepsilon_h\to 0} \left|\sum_{j\in\mathcal{N}_h(\Omega_i)\setminus\{i\}} \omega_{ij}^\beta (u_i^\beta - u_j^\beta)\right|_{1,\varepsilon_h}. \tag{3.17}$$

Hence, if $\varepsilon_h \to 0$ and $\gamma_h \to 0$, the numerator of (3.8) is smaller than 1. Therefore, $\Phi_i < 1$ and thus $\alpha^e < 1$ as ε_h and γ_h tend to 0. □

LEMMA 3.2. *(Def. 2.5) The stabilization term $B_h(\boldsymbol{u}_h; \boldsymbol{u}_h, \boldsymbol{v}_h)$ defined in (3.5) is weakly linearity-preserving, i.e. if $u_h^\beta \in \mathcal{P}_1^d(\Omega)$ for all $\beta \in C$, then*

$$\lim_{\gamma_h,\tau_h\to 0} B_h(\boldsymbol{w}_h; \boldsymbol{u}_h, \boldsymbol{v}_h) = 0 \tag{3.18}$$

Proof. It is easy to see that if the shock detector α^e is equal to 0, then the artificial diffusion, and thus the stabilization term, vanishes (see (3.5)–(3.6)). Therefore, it is sufficient to show that α^e vanishes for $u_h^\beta \in \mathcal{P}_1^d(\Omega)$ with $\beta \in C$. First, if component β of the solution is linear then its gradient is constant, hence

$$\mathbf{g}_i^\beta = \frac{1}{(\mathbb{1}, \varphi_i)}(\boldsymbol{\nabla} u_h^\beta, \varphi_i) = \frac{\boldsymbol{\nabla} u_h^\beta}{(\mathbb{1}, \varphi_i)}(\mathbb{1}, \varphi_i) = \boldsymbol{\nabla} u_h^\beta. \tag{3.19}$$

Moreover, $\boldsymbol{\nabla} u_h^\beta \cdot (\boldsymbol{x}_i - \boldsymbol{x}_j) = \mathbf{g}_i^\beta \cdot (\boldsymbol{x}_i - \boldsymbol{x}_j) = u_i^\beta - u_j^\beta$. Let $\mathcal{A} \doteq \{j \in \mathcal{N}_h(\Omega_i) : \mathbf{g}_i^\beta \cdot (\boldsymbol{x}_i - \boldsymbol{x}_j) > \tau_h\}$ and $\mathcal{B} \doteq \{j \in \mathcal{N}_h(\Omega_i) : \mathbf{g}_i^\beta \cdot (\boldsymbol{x}_i - \boldsymbol{x}_j) < -\tau_h\}$. Then, as $\tau_h \to 0$, all pairs ij will either be part of $\mathcal{A} \cup \mathcal{B}$ or $(u_i^\beta - u_j^\beta) = 0$, and the weights become

$$\omega_{ij}^\beta = \frac{1}{S_+^\beta} \quad \forall j \in \mathcal{A} \qquad \omega_{ij}^\beta = \frac{1}{S_-^\beta} \quad \forall j \in \mathcal{B}. \tag{3.20}$$

By the definition of S_+^β and S_-^β, the numerator of Φ_i^β becomes

$$\sum_{j\in\mathcal{A}} \frac{1}{S_+^\beta}(u_i^\beta - u_j^\beta) + \sum_{j\in\mathcal{B}} \frac{1}{S_-^\beta}(u_i^\beta - u_j^\beta) + \gamma_h = \sum_{j\in\mathcal{A}} \frac{S_+^\beta}{S_+^\beta} + \sum_{j\in\mathcal{B}} \frac{-S_-^\beta}{S_-^\beta} + \gamma_h = 1 - 1 + \gamma_h = \gamma_h. \quad (3.21)$$

Therefore, as $\gamma_h \to 0$, the numerator of Φ_i, and hence Φ_i will be zero. Since this is true for all $i \in \mathcal{N}_h$, it follows that $B_h(\boldsymbol{u}_h; \boldsymbol{u}_h, \boldsymbol{v}_h) = 0$ as $\tau_h, \gamma_h \to 0$ provided $u_h^\beta \in \mathcal{P}_1^d(\Omega)$, for all $\beta \in C$. ∎

Finally, we show that problem (3.13) is LED (i.e. satisfy Def. 2.4), and its scalar steady counterpart

$$\tilde{\mathsf{K}}(u_h)\mathsf{U} = \mathsf{G}, \quad (3.22)$$

satisfies the DMP in Def. 2.2.

THEOREM 3.3 (DMP). *The scalar steady discrete problem* (3.22) *satisfies the DMP in Def. 2.2 if* $\boldsymbol{g} = 0$ *in* Ω.

Proof. From [4, Th. 1] we know that if for every degree of freedom $i \in \mathcal{N}_h$, such that u_i is a local discrete extremum its value can be expressed as a convex combination of its neighbors. That means the following holds:

$$\tilde{\mathsf{K}}_{ij}(u_h) \le 0,\ \forall j \in \mathcal{N}_h(\Omega_i) \ \text{ if } \ j \neq i, \quad \text{ and } \sum_{j\in\mathcal{N}_h(\Omega_i)} \tilde{\mathsf{K}}_{ij}(u_h) = 0, \quad (3.23)$$

then u_h satisfies the local DMP in Def. 2.2. Therefore, it is enough to prove that above properties are satisfied by problem (3.22). From Lm. 3.1, we know that if u_i is a local extremum then $\alpha^e = 1$ for all elements K^e in Ω_i. Moreover, since for the scalar case $\lambda_{ij}^{\max} = |\mathsf{K}_{ij}^e|$, we see that

$$\max{}_{\sigma_h}\{\lambda_{ij}^{\max}, \lambda_{ji}^{\max}\} = \max{}_{\sigma_h}\{|\mathsf{K}_{ij}^e|, |\mathsf{K}_{ji}^e|\} \ge \mathsf{K}_{ij}^e \quad \forall j \neq i, \quad (3.24)$$

where, K_{ij}^e is the elemental stiffness matrix. Thus, $\nu_{ij}^e \ge \mathsf{K}_{ij}^e$ for any $j \neq i$. Hence, by definition of B_{ij} and $\ell(i,j)$, it is easy to see that $\tilde{\mathsf{K}}_{ij}^e = \mathsf{K}_{ij}^e + \mathsf{B}_{ij}^e \le 0$ for all $j \neq i$, which imply that this is also true for the fully assembled matrices, i.e. $\tilde{\mathsf{K}}_{ij} = \mathsf{K}_{ij} + \mathsf{B}_{ij} \le 0\ \forall j \neq i$. Further, by construction $\sum_{j\in\mathcal{N}_h(K^e)} \mathsf{B}_{ij}^e = 0$, thus $\sum_{j\in\mathcal{N}_h} \mathsf{B}_{ij} = 0$. It is well known that $\sum_{j\in\mathcal{N}_h(\Omega_i)} \mathsf{K}_{ij} = 0$, therefore $\sum_{j\in\mathcal{N}_h(\Omega_i)} \tilde{\mathsf{K}}_{ij} = 0$. Hence, it has been proven that conditions (3.23) are satisfied for any $i \in \mathcal{N}_h$, such that u_i is a local discrete extremum, and this closes the proof. ∎

COROLLARY 3.4 (Scalar LED). *The scalar counterpart of the discrete problem* (3.13) *is LED as defined in Def. 2.3 if* $\boldsymbol{g} = 0$ *in* Ω.

Proof. From [2, Th. 4.1] it is known that the scheme is LED if for every u_i that is a extremum, $\tilde{\mathsf{M}} = m_i\delta_{ij}$ and conditions (3.23) hold. From above Thrm. 3.3, we know that conditions (3.23) do hold. From Lm. 3.1 we know that if u_i is a extremum, then $\alpha^e = 1$ for all elements K^e in Ω_i. Therefore, by definition of $\tilde{\mathsf{M}}_{ij}(\boldsymbol{u}_h) = \sum_{K^e\in\mathcal{T}_h} \sum_{i,j\in\mathcal{N}_h(K^e)} (1 - \alpha^e)(\varphi_j, \varphi_i) + \alpha^e(\mathbb{1}, \varphi_i) = \sum_{K^e\in\mathcal{T}_h} \sum_{i,j\in\mathcal{N}_h(K^e)} (\mathbb{1}, \varphi_i) = m_i\delta_{ij}$. ∎

THEOREM 3.5 (LED). *The discrete problem* (3.13) *is LED as defined in Def. 2.4 if* $\boldsymbol{g} = 0$ *in* Ω *and, given an arbitrary vector* $\boldsymbol{n}$, $\lambda_j^{\max}\|\boldsymbol{n}\|$ *is the maximum eigenvalue of* $\boldsymbol{f}'(\boldsymbol{u}_j)\cdot\boldsymbol{n}$, *where* $\boldsymbol{f}'(\boldsymbol{u}_j)$ *are flux Jacobians.*

Proof. Let us assume that component $\beta \in C$ of $\boldsymbol{u}_h$ has an extremum at $\boldsymbol{x}_i$. Then, from Lm. 3.1 we know that $\alpha^e = 1$ for all elements K^e in Ω_i. Therefore, it is easy to see that in this case $\tilde{\mathsf{M}}_{ij}(\boldsymbol{u}_h) = \sum_{K^e\in\mathcal{T}_h} \sum_{i,j\in\mathcal{N}_h(K^e)} (\mathbb{1}, \varphi_i) I_{m\times m}$.

Let us now analyze how the elemental stiffness matrices are constructed. On the one hand, for the pair $i,j \in \mathcal{N}_h$ after linearizing the flux Jacobian with respect to $\boldsymbol{u}_j$, i.e. $\boldsymbol{f}'(\boldsymbol{u}_h) \simeq \boldsymbol{f}'(\boldsymbol{u}_j)$, we have that the elemental stiffness matrix can be computed as

$$\begin{aligned}
\mathsf{K}_{ij}^{e^{\beta\gamma}} \doteq & (\varphi_j \delta_{\beta\xi}, \boldsymbol{f}'(\boldsymbol{u}_j)_k^{\xi\eta} \cdot \partial_k \varphi_i \delta_{\eta\gamma})_{K^e} - (\varphi_j \delta_{\beta\xi}, n_k \cdot \boldsymbol{f}'(\boldsymbol{u}_j)_k^{\xi\eta} \varphi_i \delta_{\eta\gamma})_{K^e \cap \Gamma_{\text{out}}} && (3.25)\\
= & \delta_{\beta\xi} \boldsymbol{f}'(\boldsymbol{u}_j)_k^{\xi\eta} \delta_{\eta\gamma} (\varphi_j, \partial_k \varphi_i)_{K^e} - \delta_{\beta\xi} \boldsymbol{f}'(\boldsymbol{u}_j)_k^{\xi\eta} \delta_{\eta\gamma} (\varphi_j, n_k \varphi_i)_{K^e \cap \Gamma_{\text{out}}} && (3.26)\\
= & \boldsymbol{f}'(\boldsymbol{u}_j)_k^{\beta\gamma} [(\varphi_j, \partial_k \varphi_i)_{K^e} - (\varphi_j, n_k \varphi_i)_{K^e \cap \Gamma_{\text{out}}}] && (3.27)\\
= & \boldsymbol{f}'(\boldsymbol{u}_j)^{\beta\gamma} \cdot \boldsymbol{c}_{ij}^e, && (3.28)
\end{aligned}$$

where $\boldsymbol{c}_{ij}^e \doteq (\boldsymbol{\nabla}\varphi_j, \varphi_i)_{K^e}$. Hence, $\mathsf{K}_{ij} = \sum_e \mathsf{K}_{ij}^e = \boldsymbol{f}'(\boldsymbol{u}_j) \cdot \sum_e \boldsymbol{c}_{ij}^e$.

On the other hand, we know that given a vector, $\boldsymbol{n}$, $\rho(\boldsymbol{f}'(\boldsymbol{u}_j) \cdot \boldsymbol{n}) = \lambda_j^{\max} \|\boldsymbol{n}\|$. Thus, $\rho(\mathsf{K}_{ij}^e) = \lambda_j^{\max} \|\boldsymbol{c}_{ij}^e\|$, and $\rho(\mathsf{K}_{ij}) = \lambda_j^{\max} \|\boldsymbol{c}_{ij}\|$. Further, since $\boldsymbol{c}_{ij} = \sum_e \boldsymbol{c}_{ij}^e$, we have that

$$\sum_e \rho(\mathsf{K}_{ij}^e) = \sum_e \lambda_j^{\max} \|\boldsymbol{c}_{ij}^e\| \geq \sum_e \lambda_j^{\max} \|\boldsymbol{c}_{ij}\| = \rho(\mathsf{K}_{ij}). \tag{3.29}$$

For $j \neq i$, by definition of ν_{ij}^e it can be seen that $\sum_e \nu_{ij}^e \geq \sum_e \rho(\mathsf{K}_{ij}^e) \geq \rho(\mathsf{K}_{ij})$. Hence, since $\tilde{\mathsf{K}}_{ij} = \mathsf{K}_{ij} + \mathsf{B}_{ij}$ and $\mathsf{B}_{ij} = \sum_e \mathsf{B}_{ij}^e = \sum_e -\nu_{ij}^e I_{m\times m}$ for all $j \neq i$, this yields that the maximum eigenvalue of $\tilde{\mathsf{K}}_{ij}$ is non-positive, and this closes the proof. □

It is important to mention that the Euler equations are a particular case for the theorem presented above. All parameters used for regularizing purposes, i.e. τ_h, σ_h, γ_h, and ε_h, should be properly scaled, and must be dimensionally correct. Otherwise, spatial convergence problems might arise, as well as the inconvenience of having to tune them for each problem and particular mesh. In the present work, we scale these parameters as follows:

$$\sigma_h = \sigma |\mathbf{v}|^2 h^4 L^{-2} \quad \tau_h = \tau h^2 L^{-2} \quad \varepsilon_h = \varepsilon \quad \gamma_h = \gamma, \tag{3.30}$$

where $\mathbf{v}$ is the convection in the scalar case, and $\|\mathbf{v}\| + c$ for Euler equations. Note, that for the definition of α^e in (3.7), we do not use σ_h, but directly σ since all Φ_i are dimensionless and mesh-independent. When using uniform Cartesian meshes, the scheme becomes linearity preserving without the usage of the weights in (3.9). If these weights are not included then the parameter scaling must be redefined to $\varepsilon_h = \varepsilon h^4 L^{-4}$, and $\gamma_h = \gamma h L^{-2}$.

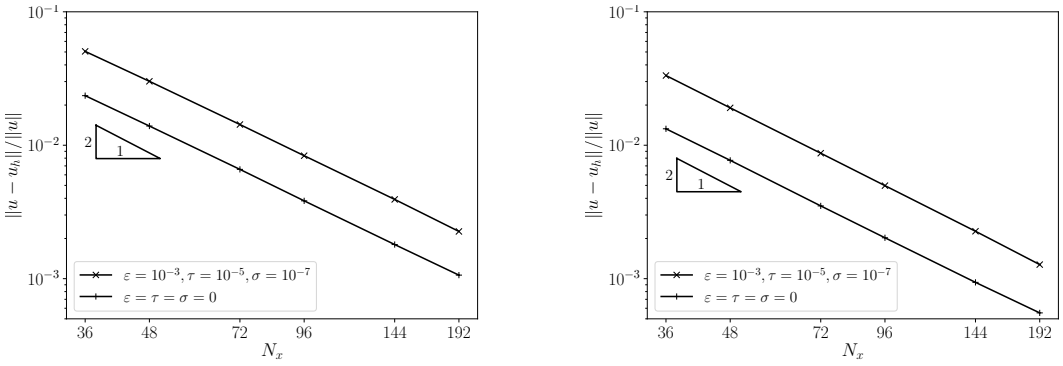
4. Numerical experiments. In this section we present some numerical experiments showing the behavior of the scheme previously introduced. First, a convergence analysis is performed in order to assess the correctness of the proposed scheme and its implementation. Then, we assess the performance of the proposed stabilization method. First, for scalar convection and finally for Euler.

4.1. Convergence analysis. To begin with, we perform a convergence analysis of the stabilized scheme proposed in the previous section. The following steady scalar pure convection problem is solved in successively refined meshes.

$$\begin{cases} \mathbf{v} \cdot \boldsymbol{\nabla} u_h &= 0 & \text{in } \Omega = [0,1]^2 \\ u_h(x,y) &= \sin\left(2\pi\left(x - \frac{y}{\tan\theta}\right)\right) & \text{on } \Gamma_{\text{in}} \end{cases}, \tag{4.1}$$

where $\mathbf{v} = (\cos\theta, \sin\theta)$, and $\theta = \pi/3$. The obvious analytical solution of above problem corresponds to the translation of the prescribed profile in the direction of the convection, $\mathbf{v}$, i.e. $u(x,y) = \sin\left(2\pi\left(x - \frac{y}{\tan\theta}\right)\right)$. It is worth mentioning that even though the continuous problem is linear, due to the stabilization terms, the discrete problem is nonlinear. We use a uniform Cartesian mesh of $N_x = \{36, 48, 72, 96, 144, 192\}$ elements in each direction.

We compare the convergence in the L_2 norm for two different choices of the stabilization parameters. On the one hand, we use $\varepsilon = \tau = \sigma = 0$, hence the nonlinear problem is not differentiable everywhere, but the scheme is linearity preserving (see Lm. 3.2). On the other hand, we choose $\varepsilon = 10^{-3}$, $\tau = 10^{-5}$, $\sigma = 10^{-7}$, $\gamma_h = 10^{-14}$, and thus the nonlinear problem is twice differentiable, but only weakly linearity preserving. For both sets, we compare the effect of choosing the parameter $q = 4$ or $q = 10$.



(a) Results for $q = 4$.

(b) Results for $q = 10$.

Fig. 4.1: Convergence of the relative error in the L_2 norm using different sets of stabilization parameters and choices of q.

Fig. 4.1 shows that regardless of the chosen stabilization parameters, the scheme achieves the theoretical convergence rates of a first order finite element scheme. However, it can be observed that as we increase ε, τ, or σ, the relative errors are increased too. For the parameter q, the behavior is analogous, its values does not affect to the convergence rate. However, the relative error is higher as q is reduced.

4.2. Effect of regularization parameters. In the previous test, we showed that the choice of the parameters has no effect on the convergence rates. However, they do affect the value of the error. In this test, we do an in-depth analysis of their influence on the error and also their effect on the computational cost. To this end, we solve the following steady pure convection problem in $\Omega \doteq [0, 0.5]^2$,

$$\begin{cases} \mathbf{v} \cdot \boldsymbol{\nabla} u_h = 0 & \boldsymbol{x} \in \Omega \\ u_h = u_D & \boldsymbol{x} \in \Gamma_{\text{in}} \end{cases}, \tag{4.2}$$

where $\mathbf{v} \doteq (y, -x)$. Its analytic solution reads,

$$u(x, y) = \begin{cases} \cos^2(\frac{10}{3}\pi(\sqrt{x^2 + y^2} - 0.4)), & \sqrt{x^2 + y^2} \in [0.075, 0.225], \\ 1, & \sqrt{x^2 + y^2} \in [0.275, 0.425], \\ 0, & \text{otherwise}, \end{cases} \tag{4.3}$$

and obviously boundary conditions are defined by $u_D(y) = u(0, y)$. As in the previous experiment, although the continuous problem is linear, the discrete problem is nonlinear.

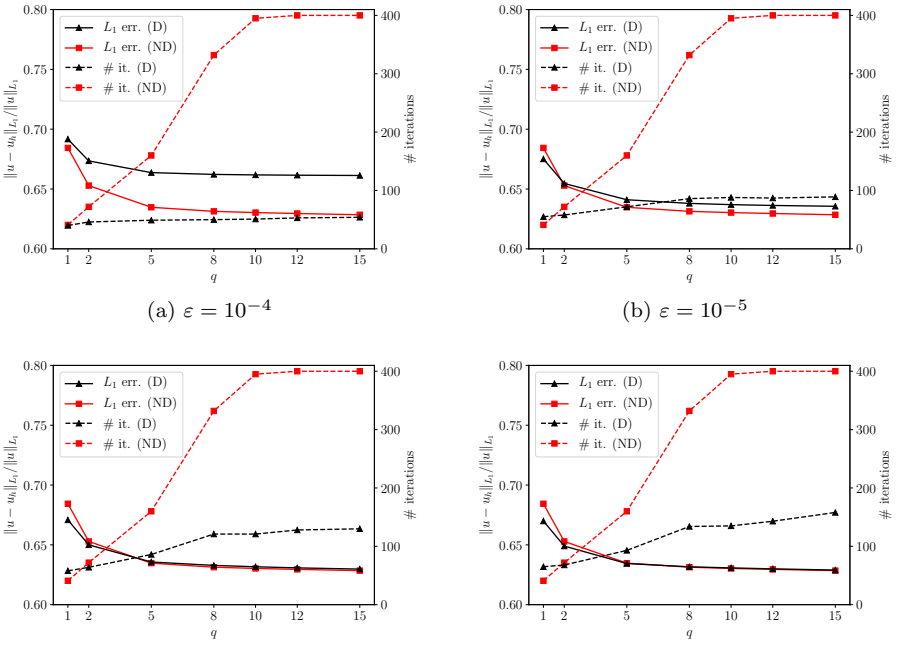
In order to perform the comparison, we use the following sets of parameter values: $q = \{1, 2, 5, 8, 10, 12, 15\}$, $\varepsilon = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$, $\tau = \varepsilon 10^{-2}$, $\sigma = \varepsilon 10^{-4}$, and $\gamma = 10^{-14}$. A 64×64 Q_1 FE mesh is used. We compare, the L_1 norm of the error at

the outflow boundary $\|u - u_h\|_{L_1(y=0)}$, and the computational cost in terms of the number of nonlinear iterations required to converge. An Anderson accelerated fixed point iteration scheme is used. We let it iterate until it satisfies:

$$\left(\frac{1}{N} \sum_{i=1}^{N} \left(\frac{\mathsf{U}_i^k - \mathsf{U}_i^{k-1}}{10^{-4}|\mathsf{U}_i^{k-1}| + 10^{-6}} \right)^2 \right)^{1/2} < 1,$$

where U is the unknown vector and N is the number of unknowns. We also impose that the residual must satisfy $\|F\|_k / \|F\|_0 < 10^{-3}$. We consider that the scheme does not converge when it reaches 400 iterations.

In Fig. 4.2 we depict, for particular values of ε, the number of iterations required to converge and the relative error $\|u - u_h\|_{L_1(y=0)} / \|u\|_{L_1(y=0)}$ as a function of q. It also shows both values for the non-differentiable scheme, i.e. $\varepsilon = \tau = \sigma = \gamma = 0$. In general, it can be observed that as q increases or as ε decreases, the error decreases, but the computational cost increases. It can also be observed that for the same level of accuracy, the differentiable scheme requires fewer iterations to converge. This becomes especially evident for very small values of the stabilization parameters, e.g. Fig. 4.2 (d).



(a) $\varepsilon = 10^{-4}$ (b) $\varepsilon = 10^{-5}$

(c) $\varepsilon = 10^{-6}$ (d) $\varepsilon = 10^{-7}$

Fig. 4.2: Comparison of the relative error at the outlet ($\|u - u_h\|_{L_1(y=0)} / \|u\|_{L_1(y=0)}$) as a function of q, for the differentiable stabilization (D) with different parameter values and the non-differentiable stabilization (ND).

4.3. Scramjet. Finally, we apply the method to the solution of the Euler equations as a proof-of-principle. In particular, we solve the $M = 3$ scramjet problem [10]. The benchmark consists of a steady flow entering in a narrowing channel with two sharp-cornered

internal obstacles. This results in multiple shock reflections. We use two different meshes, the one in Fig. 4.3 (16320 Q_1 elements), and a uniformly refined version of it (65280 Q_1 elements). Currently the exact Jacobian cannot be constructed due to implementation restrictions. Hence, we use an inexact Newton's method for solving the arising nonlinear problem. We iterate until $\left(\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\mathsf{U}_i^k-\mathsf{U}_i^{k-1}}{10^{-3}|\mathsf{U}_i^{k-1}|+10^{-3}}\right)^2\right)^{1/2} < 10^{-1}$ is satisfied, and also $\|F\|_k/\|F\|_0 < 10^{-3}$, or until it reaches 400 iterations.

The variables that we use to detect possible inadmissible values of the unknown are the density and the pressure. The stabilization parameters used are $\varepsilon = 10^{-4}$, $\tau = \varepsilon 10^{-2}$, $\sigma = \varepsilon 10^{-4}$, $\gamma = 10^{-14}$, and $q = 10$.

Figs. 4.4 and 4.5 show the results when we solve directly for the steady Euler equations. Although the tolerance on the relative residual reduction is not fully satisfied, results show a good agreement with the ones in [15,17] with well resolved shocks. Fig. 4.6 shows the relative residual evolution at every nonlinear iteration. We observe that for a coarse mesh both methods behave similarly. However, when the mesh is refined, the non-differentiable stabilization is unable to converge. As opposed, the differentiable version is able to reach the same residual reduction as for the coarse mesh.

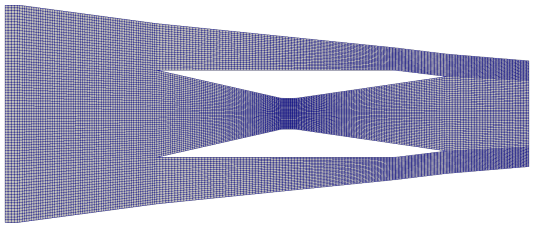


Fig. 4.3: Coarse mesh used for the scramjet test. 16320 Q_1 elements

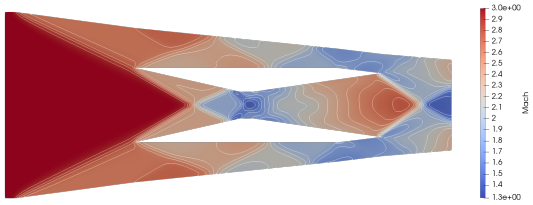


Fig. 4.4: Mach contours when the coarse mesh is used.

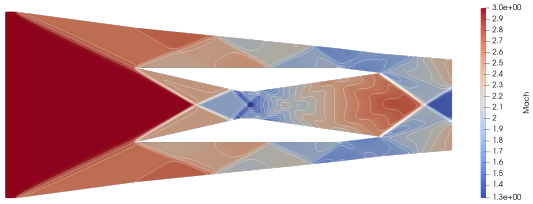


Fig. 4.5: Mach contours when the fine mesh is used.

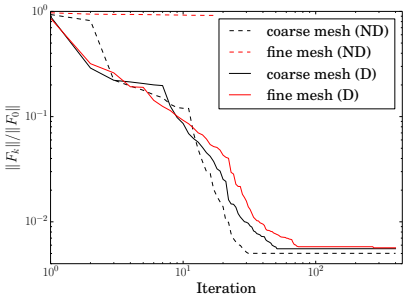


Fig. 4.6: Evolution of the relative residual at each nonlinear iteration for the differentiable and non-differentiable stabilization, and the coarse and fine meshes used.

5. Conclusion. In this work, we developed a differentiable stabilization for Euler equations and scalar transport problems. This was constructed using a differentiable shock detector, Rusanov artificial diffusion and some partial mass lumping of the time derivative term. The resulting method was shown to be LED for scalar problems and first order conservation law systems. Further, the solution of the steady scalar transport problem is proven to satisfy the local DMP. The main novelty here was the smoothing of the linearity-preserving limiter from [9] and its use as a shock detector, which improved the nonlinear convergence. Numerical results demonstrate this. However, the proposed method is only linearity-preserving in a weak sense. In particular, it is linearity-preserving when stabilization parameters $\gamma_h,\ \tau_h \to 0$. The numerical results for Euler equations show that the method is able to achieve good resolutions for problems with complex shock profiles. Future work will consist of in-depth analysis of the effect of using differentiable stabilization operators for Euler equations. In particular, if more sophisticated nonlinear solvers are used, e.g. Newton's method with an exact Jacobian.

Acknowledgments. The mesh in Fig. 4.3 is courtesy of Thomas M. Smith, the authors also acknowledge fruitful discussions with him. The authors would like to acknowledge the work of Roger P. Pawlowski and Eric C. Cyr in implementing the non-differentiable AFC stabilization. The work of Sibusiso Mabuza and John N. Shadid was partially supported by the U.S. Department of Energy, Office of Science, Office of Applied Scientific Computing Research. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. Jesús Bonilla gratefully acknowledges the support received from "la Caixa" Foundation through its PhD scholarship program. Jesús Bonilla and Santiago Badia acknowledge the financial support to CIMNE via the CERCA Programme / Generalitat de Catalunya.

REFERENCES

[1] B. Cockburn and G.E. Karniadakis and C.-W. Shu, *The development of discontinuous Galerkin methods*, in Discontinuous Galerkin methods, Springer Berlin Heidelberg, 2000, pp. 3–50.

[2] S. Badia and J. Bonilla, *Monotonicity-preserving finite element schemes based on differentiable nonlinear stabilization*, Computer Methods in Applied Mechanics and Engineering, 313 (2017), pp. 133–158.

[3] S. Badia, J. Bonilla, and A. Hierro, *Differentiable monotonicity-preserving schemes for discontinuous Galerkin methods on arbitrary meshes*, Computer Methods in Applied Mechanics and Engineering, 320 (2017), pp. 582–605.

[4] R. Codina, *A discontinuity-capturing crosswind-dissipation for the finite element solution of the*

convection-diffusion equation, Computer Methods in Applied Mechanics and Engineering, 110 (1993), pp. 325–342.

[5] G. Scovazzi and J.N. Shadid and E. Love and W.J. Rider, *A conservative nodal variational multiscale method for Lagrangian shock hydrodynamics*, Computer Methods in Applied Mechanics and Engineering, 199 (2010), pp. 3059–3100.

[6] S. Gottlieb, C.-W. Shu, and E. Tadmor, *Strong Stability-Preserving High-Order Time Discretization Methods*, SIAM Review, 43 (2001), pp. 89–112.

[7] J.L. Guermond and B. Popov and V. Tomov, *Entropy-viscosity method for the single material Euler equations in Lagrangian frame*, Computer Methods in Applied Mechanics and Engineering, 300 (2016), pp. 402–426.

[8] V. John and E. Schmeyer, *Finite element methods for time-dependent convection-diffusion-reaction equations with small diffusion*, Computer Methods in Applied Mechanics and Engineering, (2008).

[9] D. Kuzmin, S. Basting, and J. N. Shadid, *Linearity-preserving monotone local projection stabilization schemes for continuous finite elements*, Computer Methods in Applied Mechanics and Engineering, 322 (2017), pp. 23–41.

[10] D. Kuzmin, M. Matthias, and M. Gurris, *Algebraic Flux Correction II. Compressible flows*, in Flux-corrected Transport, 2012, pp. 193–238.

[11] D. Kuzmin and M. Möller, *Algebraic Flux Correction I. Scalar Conservation Laws*, in Flux-Corrected Transport, D. D. Kuzmin, P. R. Löhner, and P. D. S. Turek, eds., Scientific Computation, Springer Berlin Heidelberg, jan 2005, pp. 155–206.

[12] D. Kuzmin, M. J. Shashkov, and D. Svyatskiy, *A constrained finite element method satisfying the discrete maximum principle for anisotropic diffusion problems*, Journal of Computational Physics, 228 (2009), pp. 3448–3463.

[13] D. Kuzmin and S. Turek, *Flux Correction Tools for Finite Elements*, Journal of Computational Physics, 175 (2002), pp. 525–558.

[14] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, Cambridge, 2002.

[15] C. Lohmann and D. Kuzmin, *Synchronized flux limiting for gas dynamics variables*, Journal of Computational Physics, 326 (2016), pp. 973–990.

[16] S. Mabuza, J. Shadid, E. Cyr, D. Kuzmin, and T. Smith, *A monotone linearity preserving nodal variation limiting algorithm for continuous galerkin discretization of mhd equations*, 2019.

[17] S. Mabuza, J. N. Shadid, and D. Kuzmin, *Local bounds preserving stabilization for continuous Galerkin discretization of hyperbolic systems*, Journal of Computational Physics, 361 (2018), pp. 82–110.

A SCALABLE APPROACH FOR SOLVING STOCHASTIC INVERSE PROBLEMS BASED ON PUSH-FORWARD MEASURES AND BAYES' RULE

BRAD MARVIN *, TIM WILDEY †, AND TAN BUI-THANH ‡

Abstract. We present a scalable approach to the solution of linear and nonlinear stochastic inverse problems using the recently developed consistent Bayesian method. The use of kernel density estimation in the consistent Bayesian method can result in poor scaling with the dimensionality of the data. Our approach avoids the need for density estimation and is therefore scalable with the data dimension. We adapt a randomized dimension reduction technique developed for the classical Bayesian maximum a posteriori (MAP) point method to develop a consistent Bayesian MAP point method which scales with the data, state, and parameter dimensions. Additionally, we investigate the relationship between the consistent Bayesian and classical Bayesian techniques for linear inverse problems. We show that from a deterministic point of view, the MAP point of the consistent Bayesian method is similar to a truncated SVD approach, and that the consistent Bayesian method applies regularization only in directions uninformed by the data. We demonstrate our scalable approach on a linear inverse problem governed by an advection-diffusion equation and a nonlinear inverse problem governed by a hyperelasticity equation.

1. Introduction.

The recently developed consistent Bayesian approach for stochastic inverse problems developed in [3] solves the following problem: given data with a known distribution and a computational model which maps parameters to observables, find a distribution in the parameter space such that the push forward distribution matches the given data distribution. This is not necessarily the same setup as the classical Bayesian approach to inverse problems [2, 10, 14], since the classical Bayesian posterior is not necessarily consistent with the model and the data, i.e. the push forward of the posterior via the parameter-to-observable map does not necessarily match the distribution of our data. As noted in [3], the lone exception is the case where the push-forward of the prior is a uniform distribution. In that case, the classical and consistent Bayesian approaches give identical results. In the classical Bayesian approach, if we select a highly informative prior distribution the resulting posterior may be overly influenced by the prior and drastically inconsistent with the data. On the other hand, the consistent Bayesian posterior will always be consistent with our data under the mild assumption that the distribution of the data is absolutely continuous with respect to the push forward of the prior.

While this consistency property may be attractive for real world application when our choice of prior may be highly arbitrary, computing the consistent Bayesian posterior requires an additional complication that the classical Bayesian technique does not. The consistent Bayesian method requires estimating the push forward of the prior density via the parameter-to-observable map which can be accomplished by kernel density estimation (KDE) or parametric density estimation using a set of samples from the prior distribution. If we estimate the push forward of the prior via kernel density estimation (KDE), the push forward density will converge uniformly at a rate $\mathcal{O}\left(\frac{\ln n}{n}^{p/(d+2p)}\right)$, where d is the dimension of the data space and p is the kernel order [7]. If our data are high dimensional or computing the push forward of a sample is computationally expensive then KDE may become computationally

*The University of Texas at Austin, Institute for Computational Engineering and Sciences, Brad_marvin@utexas.edu

†Sandia National Laboratories, Optimization and Uncertainty Quantification Department, tmwilde@sandia.gov

‡The University of Texas at Austin, Institute for Computational Engineering and Sciences, Department of Aerospace Engineering and Engineering Mechanics, tanbui@ices.utexas.edu

intractable. In particular, if our parameter-to-observable map requires the solution of a partial differential equation (PDE) then computing the push forward of a single sample from the prior may require significant computational resources. If we attempt to estimate the push forward of the prior via parametric density estimation, we risk introducing significant error if the true density is not well approximated by our assumed family of distributions.

The outline of this paper is as follows. In Section 2 we review a particular scalable approach for the classical Bayesian inverse problem, in particular the maximum a posteriori (MAP) point estimation technique. This is followed by a review of the consistent Bayesian method in Section 3. In Section 3.1 we investigate how the prior influences the consistent Bayesian posterior and connect the consistent Bayesian method to the classical Bayesian technique using regularization based on truncated singular value decomposition (SVD) as described in [5, 8, 9]. In Section 4 we present a novel MAP point estimation technique for the solution of linear inverse problems with the consistent Bayesian method. Our new approach avoids the need for density estimation and utilizes a low-rank approximation method developed for the classical Bayesian method [2, 13] to achieve scalability with the dimension of the data and parameter spaces. We demonstrate our technique by solving a linear inverse problem governed by an advection-diffusion equation. In Section 5 we extend our approach to nonlinear inverse problems by introducing an iterative linearization technique and demonstrate by inverting for a parameter in a hyperelasticity equation. Finally, we summarize and conclude our discussion in Section 6.

2. A Scalable Approach for the Classical Bayesian Inverse Problem.

In this section we provide the requisite background material to support our analysis in Section 3.1 and our scalable approach for consistent Bayesian inverse problems in Sections 4 and 5. We start by summarizing scalable techniques for finding the MAP point and approximating the covariance operator for the classical Bayesian inverse problem.

2.1. Determining the MAP Point. Consider the abstract variational problem: Given λ in a Hilbert space Λ, find $u^* \in V$ such that:

$$u^*(\lambda) = \underset{u \in V}{\operatorname{argmin}} \quad \Pi(u, \lambda). \tag{2.1}$$

In addition, assume we have point observations $\{y_i\}_1^d$ for some quantities of interest (QoI) $\{B_i\}_1^d$, where each B_i maps $u^*(\lambda) \mapsto B_i(u^*(\lambda)) \in \mathbb{R}$. We define a parameter-to-observable map f to be the composition $f = B \circ u^*$.

We assume a Gaussian prior with mean μ_{pr} and covariance Γ_{pr} and a Gaussian noise model on the data with mean $\overline{\mathbf{y}}$ and covariance Γ_n. With these assumptions the maximum a posteriori (MAP) point of the classical Bayesian posterior can be found by solving the deterministic optimization problem: find $\lambda_{\text{MAP}} \in \Lambda$ which minimizes:

$$\begin{aligned} J(\lambda) &= \frac{1}{2}||\Gamma_n^{-1/2}(Bu^*(\lambda) - \overline{\mathbf{y}})||^2_{\mathbb{R}^d} + \frac{1}{2}||\Gamma_{pr}^{-1/2}(\lambda - \mu_{pr})||^2_{\mathrm{L}^2(\Omega)}. \\ &= J_{\text{misfit}}(\lambda) + J_{\text{reg}}(\lambda) \end{aligned}$$

To handle the potentially nonlinear coupling between the state u^* and parameter λ we introduce the Lagrangian form:

$$\mathcal{L}(\lambda, u, p) = \frac{1}{2}||\Gamma_n^{-1/2}(Bu - \overline{\mathbf{y}})||^2_{\mathbb{R}^d} + \frac{1}{2}||\Gamma_{pr}^{-1/2}(\lambda - \mu_{pr})||^2_{\mathrm{L}^2(\Omega)} + \mathcal{D}_p^u \Pi(u, \lambda),$$

where $\mathcal{D}_p^u$ denotes the Gâteaux derivative with respect to u in a direction p. We solve the inverse problem by seeking a point $(\lambda_{\text{MAP}}, u_{\text{MAP}}, p_{\text{MAP}})$ which satisfies the first order

optimality conditions:

$$\begin{aligned}\mathcal{D}^p_{\hat{p}}\mathcal{L} &= 0 \quad \forall \hat{p} \in V_0\\ \mathcal{D}^u_{\hat{u}}\mathcal{L} &= 0 \quad \forall \hat{u} \in V_0\\ \mathcal{D}^\lambda_{\hat{\lambda}}\mathcal{L} &= 0 \quad \forall \hat{\lambda} \in \Lambda\end{aligned}$$

In practice we set variations with respect to u and p equal to zero explicitly and perform the optimization in the parameter space to find λ_{MAP}. i.e. for each optimization iteration we solve the following system:

$$\begin{aligned}\mathcal{D}^p_{\hat{p}}\mathcal{L} &= \mathcal{D}^u_{\hat{p}}\Pi(u,\lambda) = 0 \quad \forall \hat{p} \in V_0\\ \mathcal{D}^u_{\hat{u}}\mathcal{L} &= \int_\Omega \hat{u}B^*\Gamma_n^{-1}(Bu - \overline{\mathbf{y}})\,dx + \mathcal{D}^u_{\hat{u}}\mathcal{D}^u_p\Pi(u,\lambda) = 0 \quad \forall \hat{u} \in V_0\\ \mathcal{D}^\lambda_{\hat{\lambda}}\mathcal{L} &= \int_\Omega \hat{\lambda}\Gamma_{pr}^{-1}(\lambda - \mu_{pr})\,dx + \mathcal{D}^\lambda_{\hat{\lambda}}\mathcal{D}^u_p\Pi(u,\lambda) = \int_\Omega \hat{\lambda}\mathcal{G}(\lambda)\,dx \quad \forall \hat{\lambda} \in \Lambda.\end{aligned}$$

where $\mathcal{G}$ is the gradient in the reduced space. In addition we would like to utilize Hessian information to accelerate the convergence of the optimization procedure. To find the Hessian-action we introduce the meta-Lagrangian:

$$\begin{aligned}\tilde{\mathcal{L}}(\lambda,u,p,\tilde{\lambda},\tilde{u},\tilde{p}) = \mathcal{D}^u_{\tilde{p}}\Pi(u,\lambda) &+ \int_\Omega \tilde{u}B^*\Gamma_n^{-1}(Bu - \overline{\mathbf{y}})\,dx\\ &+ \mathcal{D}^u_{\tilde{u}}\mathcal{D}^u_p\Pi(u,\lambda) + \int_\Omega \tilde{\lambda}\Gamma_{pr}^{-1}(\lambda - \mu_{pr})\,dx + \mathcal{D}^\lambda_{\tilde{\lambda}}\mathcal{D}^u_p\Pi(u,\lambda)\end{aligned}$$

The Hessian-action can be found by solving the following system:

$$\begin{aligned}\mathcal{D}^p_{\hat{p}}\tilde{\mathcal{L}} &= \mathcal{D}^u_{\tilde{u}}\mathcal{D}^u_{\hat{p}}\Pi(u,\lambda) + \mathcal{D}^\lambda_{\tilde{\lambda}}\mathcal{D}^u_{\hat{p}}\Pi(u,\lambda) = 0 \quad \forall \hat{p} \in V_0\\ \mathcal{D}^u_{\hat{u}}\tilde{\mathcal{L}} &= \int_\Omega \tilde{u}B^*\Gamma_n^{-1}B\hat{u}\,dx + \mathcal{D}^u_{\hat{u}}\mathcal{D}^u_{\tilde{p}}\Pi(u,\lambda)\\ &\quad + \mathcal{D}^u_{\hat{u}}\mathcal{D}^\lambda_{\tilde{\lambda}}\mathcal{D}^u_p\Pi(u,\lambda) + \mathcal{D}^u_{\hat{u}}\mathcal{D}^\lambda_{\tilde{\lambda}}\mathcal{D}^u_p\Pi(u,\lambda) = 0 \quad \forall \hat{u} \in V_0\\ \mathcal{D}^\lambda_{\hat{\lambda}}\tilde{\mathcal{L}} &= \int_\Omega \tilde{\lambda}\Gamma_{pr}^{-1}\hat{\lambda}\,dx + \mathcal{D}^\lambda_{\hat{\lambda}}\mathcal{D}^u_{\tilde{p}}\Pi(u,\lambda) + \mathcal{D}^\lambda_{\hat{\lambda}}\mathcal{D}^u_{\tilde{u}}\mathcal{D}^u_p\Pi(u,\lambda)\\ &\quad + \mathcal{D}^\lambda_{\hat{\lambda}}\mathcal{D}^\lambda_{\tilde{\lambda}}\mathcal{D}^u_p\Pi(u,\lambda) = \int_\Omega \hat{\lambda}\mathcal{H}(\lambda)\tilde{\lambda} \quad \forall \hat{\lambda} \in \Lambda\end{aligned}$$

2.2. A Scalable Approximation of the Covariance. After solving for the MAP point we can construct a low-rank approximation of the posterior covariance. We first introduce a linearization of the parameter-to-observable map about the MAP point:

$$f(\lambda) \approx f(\lambda_{\text{MAP}}) + F(\lambda - \lambda_{\text{MAP}}).$$

With this linearization the posterior covariance can be expressed as:

$$\Gamma_{\text{post}} = H^{-1}(\lambda_{\text{MAP}}) = (F^\top\Gamma_n^{-1}F + M\Gamma_{pr}^{-1})^{-1},$$

where M is the mass matrix from the finite element discretization of the parameter space. In practice explicitly inverting the Hessian matrix is computationally intractable so instead we use the low-rank approximation described in [2,13] which exploits the low-rank structure

of the prior-preconditioned linearized forward map $F^\top \Gamma_n^{-1} F$ for efficient inversion. To summarize this method let us first introduce the generalized SVD:

$$\begin{aligned} \Gamma_{pr} F^* \Gamma_n^{-1/2} &= U \Sigma V^\top \\ st. \quad U^* \Gamma_{pr}^{-1} U &= I. \end{aligned} \tag{2.2}$$

Note that throughout this paper we use the superscript * to denote the adjoint with respect to the parameter and data spaces. i.e. $F^* = M^{-1} F^\top$ and $U^* = U^\top M$ where M is the mass matrix from the discretization of the parameter space. With this decomposition the linearized Hessian simplifies to:

$$H = M\Gamma_{pr}^{-1} U \Sigma^2 U^* \Gamma_{pr}^{-1} + M\Gamma_{pr}^{-1} - M\Gamma_{pr}^{-1} U U^* \Gamma_{pr}^{-1},$$

and can be inverted via the Woodbury Identity:

$$\begin{aligned} \Gamma_{post} = H^{-1} &= \Gamma_{pr} M^{-1} - U G U^\top \\ G &= \operatorname{diag}\left\{ \frac{\Sigma_{ii}^2 - 1}{\Sigma_{ii}^2} \right\}. \end{aligned}$$

It was shown in [13] that this particular low-rank decomposition is optimal in the sense that it minimizes the Förstner metric and Kullback-Leibler divergence between the true and approximate posterior covariances for linear inverse problems.

In practice rather than solve the generalized SVD we solve the generalized Hermitian eigenvalue problem (GHEP):

$$\begin{aligned} \Gamma_{pr} F^* \Gamma_n^{-1} F \Gamma_{pr} &= U \Lambda U^\top \\ st. \quad U^* \Gamma_{pr}^{-1} U &= I. \end{aligned} \tag{2.3}$$

We can solve this GHEP efficiently using matrix-free randomized algorithms such as [6, 12]. The number of matrix-vector products required for these methods scales linearly with the desired rank of the decomposition. It was shown in [2] that the effective rank of the prior-preconditioned forward map depends on the information content of the data and is independent of the size of the forward problem. For PDE constrained inverse problems the dimension of the data is typically much smaller than the dimension of the parameter space. In such a case the decomposition (2.3) can be approximated with only a small number of matrix-vector products.

This Laplace approximation allows us to avoid sampling from the posterior which in turn improves scalability with respect to the parameter and state dimensions. In Section 4 and 5 we discuss how to extend this procedure for the consistent Bayesian method.

2.3. A Note on Prior Selection.

We conclude this section with a brief explanation of our prior selection. If the forward problem is governed by a PDE and the parameter resides in a function space then it is more appropriate to define the prior in the function space setting to avoid issues with mesh dependence. In addition, defining our prior in the infinite dimensional setting allows us to discretize with an appropriate finite-element method which leads to a sparse representation of our prior covariance. To this end, a Gaussian prior was selected for infinite-dimensional well-posedness and follows [2, 14]. Suppose our parameter space is L^2 on some domain Ω, then we select a prior covariance operator Γ_{pr} defined by the biharmonic operator:

$$\Gamma_{pr} = \mathcal{A}^{-2} = (\delta I + \gamma \nabla \cdot (\Theta \nabla))^{-2},$$

where the domain of $\mathcal{A}$ is given by:

$$\mathcal{D}(\mathcal{A}) = \{\lambda \in H^2(\Omega) : \gamma\Theta\nabla\lambda \cdot \mathbf{n} = 0\}.$$

and a prior mean μ_{pr} in $\mathcal{D}(\mathcal{A})$. The scalar hyperparameters δ and γ control the magnitude and length scale respectively and the tensor Θ models anisotropy in the covariance kernel.

The finite-dimensional representation of Γ_{pr} is given by:

$$\begin{aligned}
\Gamma_{pr} &= K^{-1}MK^{-1}M \quad \text{where} \\
K_{ij} &= \int_\Omega \delta\phi_i(x)\phi_j(x) + \gamma(\Theta\nabla\phi_i(x)) \cdot \nabla\phi_j(x)d\mathbf{x} \\
M_{ij} &= \int_\Omega \phi_i(x)\phi_j(x)d\mathbf{x},
\end{aligned}$$

where ϕ_i and ϕ_j are basis vectors for the finite-element representation of our parameter space. Note that the discrete form of the prior is easily invertible and that half powers $\Gamma_{pr}^{1/2}$ and $\Gamma_{pr}^{-1/2}$ are easy to compute.

3. A Consistent Bayesian Solution to the Stochastic Inverse Problem. In this section we summarize the consistent Bayesian method for stochastic inverse problems presented in [3]. The stochastic inverse problem we seek to solve can be stated as follows: given a measure of our observations denoted μ_y, construct a posterior measure μ_{post} in Λ such that the push-forward of μ_{post} via f matches the observed measure μ_y. i.e.

$$\mu_y(A) = \mu_{post}(f^{-1}(A)) \quad \forall A \in \mathcal{B}_\mathcal{D} \tag{3.1}$$

As in the classical Bayesian approach, the construction of μ_{post} is ill-posed, i.e., in general there exist many probability measures that push forward to the given measure on the observations. We introduce a prior measure μ_{prior} in order to make the problem well-defined. It was shown in [3] that, given a prior measure μ_{prior} in Λ , a posterior measure given by:

$$P_\Lambda^{\text{post}}(A) = \int_\mathcal{D} \Big(\int_{A\cap f^{-1}(d)} \pi_\Lambda^{\text{prior}}(\lambda) \frac{\pi_\mathcal{D}^y(f(\lambda))}{\pi_\mathcal{D}^{f(\text{prior})}(f(\lambda))} d\mu_{\Lambda,d}(\lambda) \Big) d\mu_\mathcal{D}(d)$$

satisfies 3.1, under the assumption that the observed measure be absolutely continuous with respect to the push-forward of the prior, i.e. $\mu_y \ll \mu_{f(\text{prior})}$. This posterior is the result of applying the disintegration theorem described in [4] to the prior measure and followed by an application of Bayes' theorem. Note that evaluation of the posterior density:

$$\pi_\Lambda^{\text{post}} = \pi_\Lambda^{\text{prior}}(\lambda) \frac{\pi_\mathcal{D}^y(f(\lambda))}{\pi_\mathcal{D}^{f(\text{prior})}(f(\lambda))}$$

requires either parametric or kernel density estimation for the push-forward of the prior in the denominator. As mentioned in Section 1, parametric density estimates can result in significant error if the true density does not reside in the parametrized family of distributions, and kernel density estimates scale poorly with the data dimension.

3.1. Connections with Data-Informed Regularization. In this section we draw a connection between the consistent Bayesian method and regularization by truncated SVD (TSVD) based approaches as described in [5, 8, 9]. For now let

us assume that the parameter-to-observable map is linear and that the prior and observed measures are Gaussian. The MAP point cost functional can be written as:

$$J(\lambda) = \frac{1}{2}||\Gamma_n^{-1/2}(F\lambda - \overline{\mathbf{y}})||^2_{\mathbb{R}^d} + \frac{1}{2}||\Gamma_{pr}^{-1/2}(\lambda - \mu_{pr})||^2_{\mathrm{L}^2(\Omega)} - \frac{1}{2}||A^{-1/2}(F\lambda - F\mu_{pr})||^2_{\mathbb{R}^d}$$

Let us consider the Hessian of the cost functional given by

$$\begin{aligned} H &= F^\top \Gamma_n^{-1} F + M\Gamma_{pr}^{-1} - F^\top (F\Gamma_{pr}F^*)^{-1}F \\ &= M\Gamma_{pr}^{-1/2}[QQ^* + I - Q(Q^*Q)^{-1}Q^*]\Gamma_{pr}^{-1/2}, \end{aligned}$$

where the common factor $Q = \Gamma_{pr}^{1/2}F^*\Gamma_n^{-1/2}$ Now denote the reduced SVD of the common factor Q as $Q = U\Sigma V^\top$. In addition insist that $U^*U = I$, $V^\top V = I$ and $VV^\top = I$. The Hessian becomes

$$H = M\Gamma_{pr}^{-1/2}(U\Sigma^2U^* + I - UU^*)\Gamma_{pr}^{-1/2}.$$

We can do some more manipulation to get

$$H = M\Gamma_{pr}^{-1/2}\tilde{U}\Big\{\begin{bmatrix}\Sigma^2 & 0\\ 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & I\end{bmatrix}\Big\}\tilde{U}^*\Gamma_{pr}^{-1/2},$$

where $\tilde{U} = [U, W]$ and W is an orthonormal extension of U such that $WW^* = I - UU^*$. If we compare this with a similarly derived expression for the classical Bayesian approach

$$H_{SB} = M\Gamma_{pr}^{-1/2}(U\Sigma^2U^* + I)\Gamma_{pr}^{-1/2} = M\Gamma_{pr}^{-1/2}\tilde{U}\Big\{\begin{bmatrix}\Sigma^2 & 0\\ 0 & 0\end{bmatrix} + \begin{bmatrix}I & 0\\ 0 & I\end{bmatrix}\Big\}\tilde{U}^*\Gamma_{pr}^{-1/2},$$

we can see that the contribution of the prior-preconditioned misfit Hessian (first term) is the same and that the regularization (second term) is different. The consistent Bayesian method applies regularization only along directions spanned by W which form the nullspace of $\Gamma_{pr}^{1/2}F^*\Gamma_n^{-1/2}$. The classical Bayesian method applies regularization in all directions, i.e., it allows the prior to influence the data-informed directions.

Now consider the posterior for the consistent Bayesian method given by

$$\begin{aligned} \Gamma_{post}^{CB} = H_{CB}^{-1} &= \Gamma_{pr}M^{-1} - \Gamma_{pr}^{1/2}UGU^*\Gamma_{pr}^{1/2}M^{-1}. \\ G &= \mathrm{diag}\Big\{\frac{\Sigma_{ii}^2 - 1}{\Sigma_{ii}^2}\Big\} \end{aligned}$$

The consistent Bayesian method updates the prior in the directions V which span the data informed subspace. Note that the posterior can be *broader* than the prior along data informed directions if $\Sigma_{ii}^2 < 1$. This happens when, for example, the prior is too confident in these data-informed directions with large noise. For the classical Bayesian method, on the other hand, the inverse of the Hessian is given by

$$\begin{aligned} \Gamma_{post}^{SB} = H_{SB}^{-1} &= \Gamma_{pr}M^{-1} - \Gamma_{pr}^{1/2}UDU^*\Gamma_{pr}^{1/2}M^{-1}. \\ D &= \mathrm{diag}\Big\{\frac{\Sigma_{ii}^2}{\Sigma_{ii}^2 + 1}\Big\} \end{aligned}$$

Notice that the posterior is always *narrower* than prior, independent of the noise level. As a result, *the solution of the classical Bayesian approach can be overconfident.*

4. A Scalable Solution for Linear Inverse Problems.
In this section we extend the scalable methodology for the classical Bayesian inverse problem summarized in Section 2 to the consistent Bayesian method for linear inverse problems. When the parameter-to-observable map is linear, the MAP point of the consistent Bayes inverse problem can be written as a deterministic optimization problem. In this case the procedure for finding the MAP point follows the classical Bayesian approach using low-rank approximation. When the parameter-to-observable map is nonlinear, the addition of the push forward of the prior prevents us from writing the MAP point as the solution of a deterministic optimization problem. However, The MAP point can still be approximated through a partial linearization approach described in section 5.

Consider the case of a linear parameter-to-observable map and denote the linear map as $f(\lambda) = F\lambda$. Assuming a Gaussian prior and observed density, the MAP point of the consistent Bayesian posterior can be written as a the solution of a deterministic optimization: find λ_{MAP} which minimizes the cost functional:

$$J(\lambda) = \frac{1}{2}||\Gamma_n^{-1/2}(F\lambda - \overline{\mathbf{y}})||^2_{\mathbb{R}^d} + \frac{1}{2}||\Gamma_{pr}^{-1/2}(\lambda - \mu_{pr})||^2_{\mathrm{L}^2(\Omega)} - \frac{1}{2}||A^{-1/2}(F\lambda - F\mu_{pr})||^2_{\mathbb{R}^d} \tag{4.1}$$

where $A = (F\Gamma_{pr}F^*)$. If we use the prior described in Section 2.3 then the self-adjoint Hessian operator is given by:

$$\mathcal{H} = F^*\Gamma_n^{-1}F + \Gamma_{pr}^{-1} - F^*(F\Gamma_{pr}F^*)^{-1}F,$$

where F^* is the adjoint of the forward operator. After discretization the resultant symmetric Hessian matrix is given by:

$$H = F^\top\Gamma_n^{-1}F + M\Gamma_{pr}^{-1} - F^\top(F\Gamma_{pr}F^*)^{-1}F.$$

The last term in the Hessian, introduced by the push forward of the prior, greatly increases the cost of each Hessian vector product. Rather than computing the Hessian action exactly we instead use the low-rank decomposition described in Section 2. When we apply the generalized SVD (2.2), the Hessian matrix simplifies to:

$$H = M\Gamma_{pr}^{-1} + M\Gamma_{pr}^{-1}U(\Sigma^2 - I)U^*\Gamma_{pr}^{-1}$$

which can be inverted via the Woodbury matrix identity.

$$H^{-1} = \Gamma_{post} = \Gamma_{pr}M^{-1} - UGU^\top \tag{4.2}$$
$$G = \text{diag}\left\{\frac{\Sigma_{ii}^2 - 1}{\Sigma_{ii}^2}\right\}$$

The action of the approximate inverse Hessian can be computed at relatively little cost if the rank of the decomposition is small. We minimize the cost functional 4.1 using an inexact-Newton method. The details of the optimization are presented in the next section.

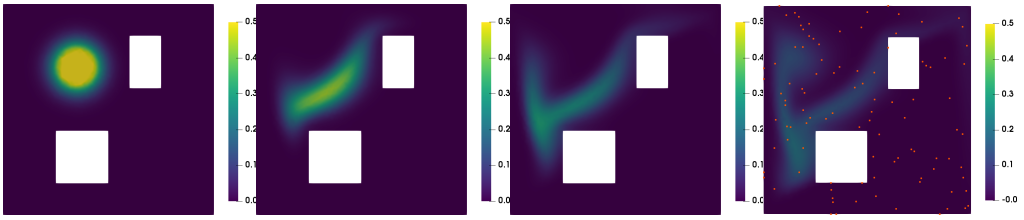
4.1. Application to Linear Advection-Diffusion.

Consider the advection-diffusion equation

$$\frac{\partial u}{\partial t} - \kappa\Delta u + \mathbf{v}\cdot\nabla u = 0 \quad \forall (x,t) \in \Omega \times (0,T),$$

$$\nabla u \cdot \mathbf{n} = 0 \quad \forall (x,t) \in \partial\Omega \times (0,T),$$
$$u(\cdot,0) = \lambda \quad \forall x \in \Omega,$$

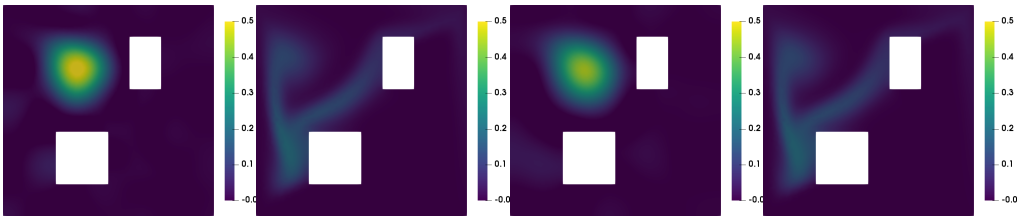
where κ is a diffusion coefficient and $\mathbf{v}$ is a non time-varying velocity field. Our problem is as follows: given 100 pointwise observations of u at $t = 3$ with added noise, infer the initial condition λ. Figure 4.1 shows the propagation of u from the initial condition to the final time. The dimension of both the discretized parameter and state spaces is 7863. We solve the inverse problem with both the classical Bayesian approach and the consistent



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$

Fig. 4.1: λ_{true} and observations at $t = 3$

Bayesian approach in Python using FEniCS [1,11] and hIPPYlib [15]. The optimization was performed using inexact-Newton globalized with backtracking line search. The Newton step was computed using the approximate inverse Hessian given by (4.2). Figure 4.2 summarizes these results. Note that the classical and consistent Bayesian methods produced similar λ_{MAP}. We take advantage of this similarity for the nonlinear approach presented in Section 5.



(a) Consistent λ_{MAP} (b) Consistent $F\lambda_{MAP}$ (c) Classical λ_{MAP} (d) Classical $F\lambda_{MAP}$

Fig. 4.2: Linear Inversion Results

The low-rank decomposition described in section 4 was used in both methods. Based on the rapid eigenvalue decay of the prior-preconditioned map (shown in Figure 4.3), 50 eigenpairs were used in the decomposition.

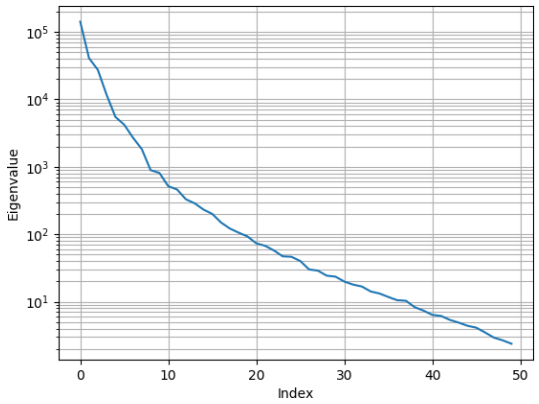


Fig. 4.3: Generalized Eigenvalues of $(H_{mis}, M\Gamma_{pr}^{-1})$

5. A Scalable Iterative Approach for Nonlinear Inverse Problems.
Now let us examine the case of a nonlinear parameter-to-observable map. If we try to naively apply MAP point estimation we run into a major obstacle. That is, the consistent Bayesian MAP point cannot be written as the solution of a deterministic optimization problem due to the addition of the nonlinear push forward of the prior. In addition, applying the linearization strategy used in Section 2 requires prior knowledge of the MAP point which we do not know. Our proposed solution is to linearize the consistent Bayesian regularization around the classical Bayesian MAP point, i.e., we first solve the classical Bayesian MAP point problem, which gives us $\tilde{\lambda}_{\text{MAP}}$ that is, as we saw in Section 4.1, close to the consistent Bayesian MAP point. Next we linearize the parameter-to-observable MAP about $\tilde{\lambda}_{\text{MAP}}$:

$$f(\lambda) \approx f(\tilde{\lambda}_{\text{MAP}}) + F(\lambda - \tilde{\lambda}_{\text{MAP}}).$$

We use this linearization only to construct the push forward of the prior, leaving the misfit term J_{misfit} fully nonlinear. The result is the following cost functional:

$$\begin{aligned} J_{PL}(\lambda) &= \frac{1}{2}||\Gamma_n^{-1/2}(f(\lambda) - \overline{\mathbf{y}})||_{\mathbb{R}^d} + \frac{1}{2}||R^{1/2}(\lambda - \mu_{pr})||_{\text{L}^2(\Omega)}, \\ R &= M\Gamma_{pr}^{-1} - F^\top(F\Gamma_{pr}F^*)^{-1}F \end{aligned}$$

Rather than construct F explicitly, we use the action of $H_{mis}(\tilde{\lambda}_{MAP})$ to solve the GHEP (2.3) and use the generalized eigenpairs to construct a low-rank version of the regularization matrix. From there we use the same optimization strategy described in Section 2. Algorithm 5.1 summarizes this procedure.

5.1. Application to Hypereleasticity.
In this section we apply our partial linearization approach to a particular nonlinear inverse problem. Consider the displacement of a compressible Mooney-Rivlin solid which can be found by solving the following nonlinear minimization problem:

$$\begin{aligned} \min_{u \in V} \quad \Pi(u) &= \int_\Omega \psi(u)\, dx - \int_\Omega B \cdot u\, dx - \int_{\partial\Omega} T \cdot u\, dS, \\ \psi(u) &= C_1(\overline{I}_1 - 3) + C_2(\overline{I}_2 - 3) + C_3(J-1)^2, \end{aligned}$$

Algorithm 1: Partial Linearization Approach

Invert for classical Bayesian $\tilde{\lambda}_{\text{MAP}}$. Set $\lambda_{\text{MAP}} = \tilde{\lambda}_{\text{MAP}}$
while $J_{misfit}(\lambda) > \epsilon$ **do**
    Find dominant eigenpairs:

$$(\Gamma_{pr}M^{-1})H_{mis}(\lambda_{\text{MAP}})(\Gamma_{pr}M^{-1}) = U\Lambda U^{\top} \quad s.t. \quad U^{*}\Gamma_{pr}^{-1}U = I$$

    Construct low-rank regularization:

$$R = M\Gamma_{pr}^{-1} - M\Gamma_{pr}^{-1}UU^{*}\Gamma_{pr}^{-1}$$

    Invert for new λ_{MAP} using $R^{1/2}$ as Tikhonov Regularization
end
Construct low-rank posterior covariance:

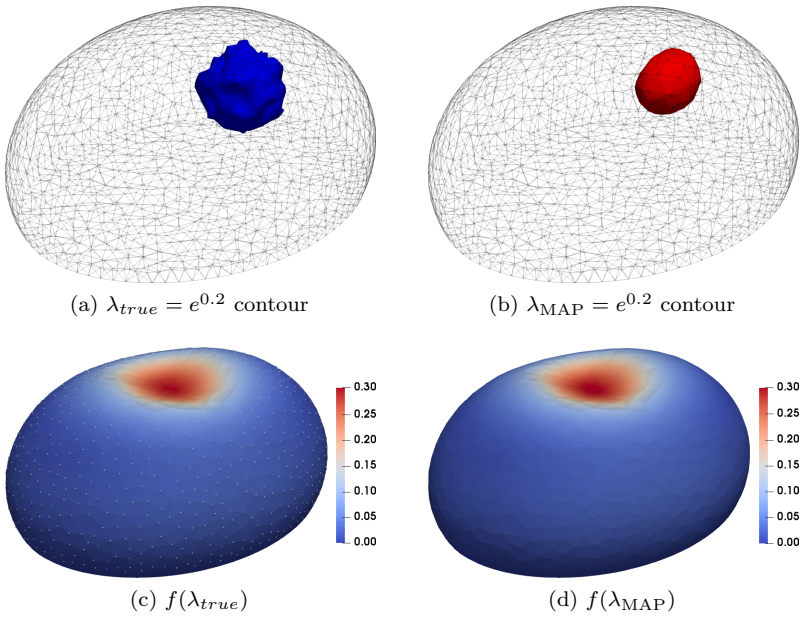
$$\Gamma_{post} = \Gamma_{pr}M^{-1} - UGU^{\top}$$
$$G = \text{diag}\Big(\frac{\Lambda_{ii} - 1}{\Lambda_{ii}}\Big)$$

Fig. 5.1: Partial Linearization Approach

$$\overline{I}_1 = J^{-2/3}\,\text{tr}(C), \quad \overline{I}_2 = J^{-4/3}\big(\frac{1}{2}\,\text{tr}(C)^2 - \frac{1}{2}\,\text{tr}(C^2)\big),$$
$$J = \det(F), \quad C = F^{\top}F, \text{ and } F = I + \nabla u.$$

Given observations of the boundary displacement we wish to infer the first coefficient in the strain-energy density C_1. In addition we wish to enforce positivity by setting $\lambda = \log(C_1)$. The true coefficient is piecewise constant with value of 0.3 inside a small inclusion and 0.1 everywhere else in the domain. Our primary interest is to detect the location of the inclusion without prior knowledge of its existence.

We use Algorithm 5.1 to solve for the consistent Bayes MAP point. The dimension of the discretized parameter and state spaces is 36905 and 14487 respectively. The optimization was carried out with an inexact-Newton conjugate gradient (CG) algorithm globalized with a backtracking line search. Figure 5.2 shows the results of the optimization.



(a) $\lambda_{true} = e^{0.2}$ contour (b) $\lambda_{\text{MAP}} = e^{0.2}$ contour

(c) $f(\lambda_{true})$ (d) $f(\lambda_{\text{MAP}})$

Fig. 5.2: Inversion results

Only 60 eigenpairs were needed for the low-rank decomposition despite the fact that the dimension of the data space is relatively high (2778). Figure 5.3 shows the rapid decay in the eigenvalues of the prior-preconditioned misfit Hessian.

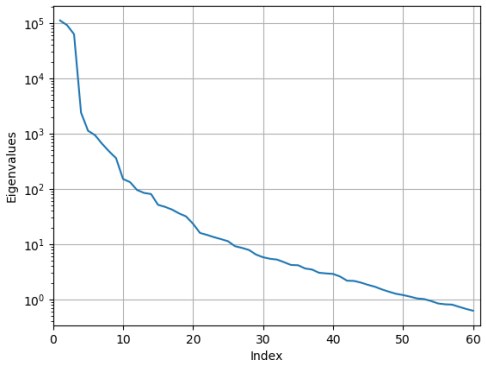


Fig. 5.3: Generalized Eigenvalues of $(H_{mis}(\tilde{\lambda}_{map}), M\Gamma_{pr}^{-1})$

6. Conclusion.

In this report we presented novel techniques for solving linear and nonlinear stochastic inverse problems with consistent Bayesian inference. These techniques avoid the need for kernel density estimation, thereby greatly improving scalability with respect to the data dimension. In addition a connection was drawn between the consistent Bayesian method and the classical Bayesian method with regularization by truncated SVD. We presented a MAP point estimation technique for linear inverse problems with consistent Bayesian inference

that utilized a low-rank decomposition of the parameter-to-observable map to reduce the cost of Hessian based optimization. This approach was demonstrated for a PDE-constrained linear inverse problem with a parameter dimension of 7863 and data dimension of 100. For nonlinear inverse problems with consistent Bayesian inference, the MAP point is no longer a solution of a deterministic optimization problem. In order to overcome this issue, we constructed an approximation of the consistent Bayesian MAP point which could be written as a deterministic optimization problem. This approach was demonstrated with a PDE-constrained nonlinear inverse problem with a parameter dimension of 36905 and data dimension of 2778.

REFERENCES

[1] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, *The fenics project version 1.5*, Archive of Numerical Software, 3 (2015).

[2] T. Bui-Thanh, O. Ghattas, J. Martin, and G. Stadler, *A Computational Framework for Infinite-Dimensional Bayesian Inverse Problems Part I: The Linearized Case, with Application to Global Seismic Inversion*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2494–A2523.

[3] T. Butler, J. Jakeman, and T. Wildey, *Combining Push-Forward Measures and Bayes' Rule to Construct Consistent Solutions to Stochastic Inverse Problems*, SIAM Journal on Scientific Computing, 40 (2018), pp. A984–A1011.

[4] J. T. Chang and D. Pollard, *Conditioning as disintegration*, Statistica Neerlandica, 51, pp. 287–317.

[5] L. Dykes and L. Reichel, *Simplified GSVD computations for the solution of linear discrete ill-posed problems*, Journal of Computational and Applied Mathematics, 255 (2014), pp. 15 – 27.

[6] N. Halko, P. G. Martinsson, and J. Tropp, *Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions*, SIAM Review, 53 (2011), pp. 217–288.

[7] B. E. Hansen, *Uniform convergence rates for kernel estimation with dependent data*, Econometric Theory, 24 (2008), p. 726–748.

[8] P. C. Hansen, *The truncated SVD as a method for regularization*, BIT, 27 (1987).

[9] P. C. Hansen, T. Sekii, and H. Shibahashi, *The Modified Truncated SVD Method for Regularization in General Form*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 1142–9.

[10] M. C. Kennedy and A. O'Hagan, *Bayesian calibration of computer models*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63 (2001), pp. 425–464.

[11] A. Logg, K.-A. Mardal, G. N. Wells, et al., *Automated Solution of Differential Equations by the Finite Element Method*, Springer, 2012.

[12] A. K. Saibaba, L. Jonghyun, and P. K. Kitanidis, *Randomized algorithms for generalized Hermitian eigenvalue problems with application to computing Karhunen–Loève expansion*, Numerical Linear Algebra with Applications, 23 (2010), pp. 314–339.

[13] A. Spantini, A. Solonen, T. Cui, J. Martin, L. Tenorio, and Y. Marzouk, *Optimal Low-rank Approximations of Bayesian Linear Inverse Problems*, SIAM Journal on Scientific Computing, 37 (2015), pp. A2451–A2487.

[14] A. M. Stuart, *Inverse problems: A Bayesian perspective*, Acta Numerica, 19 (2010), p. 451–559.

[15] U. Villa, N. Petra, and O. Ghattas, *hIPPYlib: an Extensible Software Framework for Large-scale Deterministic and Bayesian Inversion*, (2016).

COMPARING MCMC DIAGNOSTICS AND STOPPING RULES

NATHAN L. ROBERTSON*, MOHAMMAD KHALIL†, AND BRIAN M. ADAMS‡

Abstract. Markov chain Monte Carlo (MCMC) sampling is an important tool in Bayesian analysis for conducting inference about parameters of interest. The distributions of interest are often challenging to sample from and sometimes MCMC samplers fail to accurately capture the information of interest due to either a failure of the sampler to capture the distribution of interest or not enough samples being taken. Several tools to diagnose these problems exist in various settings and types of MCMC samplers. This work compares several tools designed to work for general MCMC samplers to determine recommendations for diagnostic implementation in the Dakota software developed at Sandia National Laboratories.

1. Introduction. Markov chain Monte Carlo (MCMC) techniques are tools to draw correlated samples from an approximate probability distribution of interest. These techniques are often utilized in the Bayesian setting when approximating quantities from a posterior distribution which may be intractable to construct directly. Bayesian techniques enjoy great flexibility in model specification, the ability to specify prior information to guide an analysis and the ability to output distributions. One application is model calibration for quantifying uncertainty in input parameters such as used in the Dakota software [1] developed at Sandia National Laboratories. This study is motivated by the goal of determining which measures of quality to convey to users of Dakota when MCMC simulations are performed.

Bayesian methods combine a prior belief about some parameters with information obtained from data to update a belief about the parameters. Take $f(\theta)$ to be the prior distribution, that is some distribution which represents the current knowledge of θ, and $f(D|\theta)$, the likelihood of the data given the parameter values. The posterior distribution $f(\theta|D)$ represents the new belief about θ given what we have observed

$$f(\theta|D) = \frac{f(D|\theta)f(\theta)}{\int f(D|\theta)f(\theta)d\theta}. \tag{1.1}$$

This approach differs from the frequentist approach which focuses primarily on the likelihood. In practice Bayesian models are often complicated versions of (1.1) with high dimension and dependencies between parameters.

MCMC works by constructing a Markov chain with invariant distribution equal to the distribution of interest, which in the Bayesian setting is the posterior distribution. The Markov chain is used to construct a sample which will be approximately distributed according to the invariant distribution. One may then use this sample to analyze the distribution, however there are challenges. The samples will be correlated and as such inference must take this correlation into account. Some questions this raises are "Does the sample accurately reflect the distribution of interest?", "How good is the estimate?" and "How many samples are enough?"

Some methods that attempt to answer these questions are the Gelman-Rubin diagnostic [8], Geweke diagnostic [9], Heidelberger-Welch diagnostic [12], Raftery-Lewis diagnostic [18] and relative metric fixed-volume sequential stopping rules [20] which will subsequently be referred to as sequential stopping rules. Other diagnostics exist to address these questions and were studied by Cowles and Carlin [4] who concluded that each was limited in the scope of which problems with a sampler it could detect. However, many are graphical tools or only

*University of California, Riverside, nathan.robertson@email.ucr.edu
†Sandia National Laboratories, mkhalil@sandia.gov
‡Sandia National Laboratories, briadam@sandia.gov

apply to certain types of MCMC samplers such as the Gibbs sampler and were thus not considered.

This analysis will primarily focus on addressing the quality of estimates and how informative each considered method is to determining how many samples should be drawn. Sequential stopping rules represent advances in MCMC theory while the other methods predate the influential work of Meyn and Tweedie [15] which laid the foundation for current MCMC output analysis. Sequential stopping rules, Raftery-Lewis diagnostic and Heidelberger-Welch diagnostic all attempt to place an error bound on the quantity of interest in addition to determining how many samples is enough by terminating the simulation once this error drops below some threshold. The Gelman-Rubin diagnostic runs multiple chains starting from "overdispersed" values and monitors the between chain variance compared to the within chain variance. Once the values become sufficiently close the chain may be considered "converged". The Geweke diagnostic compares the means of values early in the chain compared to the second half of the chain and considers the chain "converged" once the values are sufficiently close. Raftery-Lewis and sequential stopping rules consider the estimation of quantiles while the other methods and sequential stopping rules can consider the estimation of expectations.

This report is constructed as follows. First, an overview of MCMC theory is presented to lay the foundations for the various diagnostics and what they attempt to address. Next, a brief overview of the theory for each diagnostic is given as well as a list of pros and cons. A simulation study is then performed for which a summary of outcomes are included. Finally, recommendations for when to consider each diagnostic are offered as well as some concluding remarks.

2. MCMC Theory. Consider a probability distribution π over the support $X \in \mathbb{R}^d$. We may be interested in estimating some quantity of interest θ defined as for a function $g : \mathsf{X} \to \mathbb{R}^p$

$$\Theta = E_\pi[g(X)] = \int_{\mathsf{X}} g(X)\pi(dx). \tag{2.1}$$

The flexible nature of g allows for Θ to represent various quantities of interest such as moments or probabilities. To evaluate Θ we may turn to MCMC sampling.

Consider a Markov transition kernel $P(x, dy)$ on a general state space $(\mathsf{X}, \mathcal{B}(\mathsf{X}))$ with associated Harris Ergodic Markov chain $X = \{X_i : i = 0, 1, ...\}$ as defined in [15] and invariant distribution π. Let the $n-$step Markov transition kernel be denoted $P^n(x, dy)$ for $n \in \mathbb{N}$. Then for $x \in X$ and $A \in \mathcal{B}(X)$

$$P^n(x, A) = P(X_{i+n} \in A | X_i = x), \tag{2.2}$$

and for an initial probability measure $\lambda(\cdot)$

$$P^n(\lambda, A) = \int_{\mathsf{X}} P^n(x, A)\lambda(dx). \tag{2.3}$$

The convergence of interest is

$$||P^n(\lambda, \cdot) - \pi(\cdot)|| \overset{n\to\infty}{\longrightarrow} 0, \tag{2.4}$$

where $||\cdot||$ is the total variation norm [15]. The rate of this convergence measures how quickly the Markov transition kernel approaches the invariant distribution and is usually quantified by a bound such as $M(x)\gamma(n)$ where different forms of the functions M and γ indicate the speed of convergence. When $\gamma(n) = t^n$ for some $t < 1$ the chain is referred

to as being geometrically ergodic. The chain is uniformly ergodic if $\gamma(n) = t^n$ and M is bounded. The chain is polynomial ergodic of order m for $m \geq 1$ if $\gamma(n) = n^{-m}$. A discussion of results and conditions for the various forms of ergodicity are also provided by Meyn and Tweedie [15].

Concerns about convergence of an MCMC sampler are referring to the convergence stated in equation (2.4). Several diagnostics aim to diagnose whether this convergence has occurred. Caution should be used with those that claim convergence of a chain as occurring after some number of samples are drawn. Each sample after "convergence" is still a correlated draw from the same Markov chain with stationary transition probabilities and as such should not be discarded for reasons of convergence.

2.1. Estimation of Expectations and Markov chain Central Limit Theorem. Θ may be estimated with

$$\hat{\Theta} = \frac{1}{n}\sum_{i=1}^{n} g(X_i). \tag{2.5}$$

Under suitable regularity conditions, which are generally a form of ergodicity and a moment condition as discussed in [13, 14, 20], there exists a Markov chain central limit theorem (CLT) such that

$$\sqrt{n}(\hat{\Theta} - \Theta) \to N_p(0, \Sigma), \tag{2.6}$$

where $\Sigma = Cov(g(X_1), g(X_1)) + \sum_{j=1}^{\infty}\left[Cov(g(X_1), g(X_{1+j})) + Cov(g(X_1), g(X_{1+j}))'\right]$. Estimation of Σ is challenging in that it includes the covariance of the distribution of interest as well as the lag covariance between draws from the Markov chain. Some techniques which address this estimation include spectral variance estimators, batch means estimators and regenerative sampling as discussed in [19–21] respectively. The implementations of the methods we consider in this study will use batch means estimators or spectral variance estimators. In general batch means estimators are much faster to compute but converge more slowly than spectral variance estimators. Batch means estimators calculate the variance between the means of batches while spectral variance estimators calculate a weighted sum of estimates of autocovariances over lag windows.

Several diagnostics considered will be based on quantifying the variability in $\hat{\Theta}$ using (2.6). Additionally, the diagonal entries of Σ may be used to compute Monte Carlo standard errors for each dimension of Θ.

2.2. Estimation of Quantiles and Markov chain Central Limit Theorem. Let $W \sim \pi$ and $h : \mathsf{X} \to \mathbb{R}$. Then for $V = h(W)$ let F_V and f_V be continuous distribution and density functions of V. The q^{th} quantile of V is defined as

$$\xi_q = F_V^{-1}(q) = \inf\{v : F_V(v) = q\}. \tag{2.7}$$

This definition allows for quantiles of the marginal distributions of π. Applications may include finding credible intervals for quantities of interest.

A consistent estimator for ξ_q is

$$\hat{\xi}_q = h(X)_{\lceil nq \rceil : n}, \tag{2.8}$$

the nq^{th} order statistic of $h(X)$.

Under regularity conditions discussed in [5] there exists a CLT such that

$$\sqrt{n}(\hat{\xi}_q - \xi_q) \to N\left(0, \frac{\sigma_q^2}{(f_V(\xi_q))^2}\right), \tag{2.9}$$

where

$$\sigma_q^2 = Var(h(X)_1) + 2\sum_{j=1}^{\infty} Cov(h(X)_1, h(X)_{1+j}). \tag{2.10}$$

Results for a quantile Markov chain CLT are used in the same way as the version for expectations as detailed in Section 2.1.

2.3. Burn-in. A common practice with MCMC sampling is to discard an early portion of the chain or burn-in. Theoretical justifications for discarding burn-in are somewhat missing. In practice it can be used to start the analysis in a region of higher probability for the distribution of interest or to reduce starting value bias. In theory these are unnecessary as the starting value bias is asymptotically zero and the regions of low probability will eventually be visited in a chain of infinite length. However, these conditions are not always reasonable and burn-in may be one method to improve the quality of estimates when samples are limited. Another way to address these concerns without discarding samples is to be more selective with the choice of starting values such as using a maximum a posteriori (MAP) estimate as a starting value. In some cases burn-in can be a more efficient choice and should be looked into on a case-by-case basis.

3. Diagnostics Considered. For this analysis four diagnostics that are commonly used are examined as well as sequential stopping rules which represent a newer application of MCMC theory. The diagnostics and what they consider the estimation of are:

1. Geweke Diagnostic
2. Gelman-Rubin Diagnostic
3. Raftery-Lewis Diagnostic (Quantiles)
4. Sequential Stopping Rules (Expectations and Quantiles)
5. Heidelberger-Welch Diagnostic (Expectations)

3.1. Geweke Diagnostic. Geweke's diagnostic [9] compares a mean from early samples in a Markov chain with a mean from the later samples in the Markov chain. The idea is that in a converging chain the estimated mean of some function of the chain should take a similar value over different portions of the sample. A test is conducted to determine if there is a significant difference between the means in which case the chain may be failing to converge.

Take Θ as defined in equation (2.1) with $d = 1$. For n samples from the Markov chain X, let $A \in \mathbb{N}$ and $B \in \mathbb{N}$ be two values such that $(A + B)/n < 1$. Let

$$\hat{\theta}_A = \frac{1}{A}\sum_{i=1}^{A} g(X_i), \tag{3.1}$$

and

$$\hat{\theta}_B = \frac{1}{B}\sum_{i=n-B+1}^{n} g(X_i), \tag{3.2}$$

be the estimates of θ based on the first A and final B samples respectively. Let $\hat{S}^A(0)$ and $\hat{S}^B(0)$ be consistent spectral variance estimates of $\{g(X_i), i = 1, ..., A\}$ and $\{g(X_i), i = n - B + 1, ..., n\}$. Consistency results about univariate spectral variance estimators are available in [7]. Then when a CLT holds,

$$\frac{\hat{\theta}_A - \hat{\theta}_B}{\frac{\hat{S}^A(0)}{A} + \frac{\hat{S}^B(0)}{B}} \overset{n\to\infty}{\longrightarrow} N(0, 1). \tag{3.3}$$

A hypothesis test may then be conducted with large values providing evidence that the mean estimate has not stabilized.

While the theory allows for many choices of A and B, Geweke suggested the use of setting $A = .1n$ and $B = .5n$.

Pros:

- Checks that the estimate of θ stabilizes

Cons:

- Does not quantify the error of estimation
- Requires choice of rejection region
- May disregard large portions of the chain in diagnosing convergence
- Requires choice of which proportions of the chain to use
- Without correction appears heavily influenced by starting values

3.2. Gelman-Rubin Diagnostic. The Gelman and Rubin diagnostic [8] attempts to diagnose convergence as when the variances across m independent chains appear to stabilize to an overall variance after starting at "overdispersed" starting values. Consider a posterior mean x of interest. Simulate m independent chains X_i of length $2n$ and discard the first n samples of each chain. Using the remaining n samples, the variance between sequence means, B/n, is

$$\frac{B}{n} = \frac{1}{m-1}\sum_{i=1}^{m}(\bar{x}_{i.} - \bar{x}_{..})^2, \tag{3.4}$$

where $\bar{x}_{i.}$ is the mean of sequence i and $\bar{x}_{..}$ is the overall mean across all the chains. Let s_i^2 be the within-sequence variance estimate for sequence i. Then the average within-sequence variance, W, is

$$W = \frac{1}{m}\sum_{i=1}^{m} s_i^2. \tag{3.5}$$

The overestimate of the target variance is

$$\hat{\sigma}^2 = \frac{n-1}{n}W + \frac{1}{n}B, \tag{3.6}$$

and is asymptotically unbiased. Under the assumption that x is Normal, a t distribution may be used as a conservative estimate with center $\bar{x}_{..}$, scale

$$\hat{V} = \hat{\sigma}^2 + \frac{B}{nm} \tag{3.7}$$

and degrees of freedom $df = 2\hat{V}^2/\hat{var}(\hat{V})$. The potential scale reduction factor (PSRF) is defined as

$$\sqrt{\hat{R}} = \sqrt{\frac{\hat{V}}{W}\frac{df}{df-2}}. \tag{3.8}$$

A large scale reduction factor indicates that inference about x may be improved by taking additional samples.

This procedure is suggested to be run for each quantity of interest until each PSRF is sufficiently small. Brooks and Gelman [3] develop a refined version

$$\sqrt{\hat{R}_C} = \sqrt{\frac{df+3}{df+1}\frac{\hat{V}}{W}}, \tag{3.9}$$

and develop a multivariate extension.

Pros:

- Takes advantage of parallel computing
- Can detect prescence of a local mode

Cons:

- Requires multiple chains (and requires choice of how many chains)
- Discards more samples than probably reasonable
- Requires "overdispersed" starting values
- Does not control for quality of estimation
- Does not inherently address question of how many samples are required

3.3. Raftery-Lewis Diagnostic. Raftery and Lewis developed a procedure for estimating a quantile based on two-state Markov chain theory [18]. Consider a $\phi-$mixing Markov chain X as described in [2], quantile of interest ξ_q, an error tolerance r and a probability s. The goal is to estimate ξ_q and determine a total number of samples N and an amount of burn-in samples M such that

$$P(|\hat{P}(g(X) < \xi_q) - q| < r) = s, \tag{3.10}$$

where

$$\hat{P}(g(X) < \xi_q) = \frac{1}{N-M} \sum_{i=M+1}^{N} I(g(X_i) < \hat{\xi}_q). \tag{3.11}$$

Using notation from (2.2), this is setting s equal to the probability the difference between the estimated CDF of V evaluated at the q^{th} quantile and q is less than r.

Letting $Z_i = I(g(X_i) < \xi_q)$ and $Z_i^{(k)} = Z_{1+(i-1)k}$ yields a binary process $\{Z_i^{(k)}\}$ which will approximately be a Markov chain for sufficiently large k. If Z_i is ergodic, then $\{Z_i^{(k)}\}$ is close to a first order Markov chain. This allows for the use of two-state Markov chain theory with transition probabilities α and β. Taking k as a thinning rate and ϵ as an error tolerance between $P(Z_m^{(k)} = i|Z_0^{(k)})$ and the equilibrium distribution, the corresponding burn-in is

$$M = k\frac{\log\left(\frac{\epsilon(\alpha+\beta)}{\max(\alpha,\beta)}\right)}{\log(1-\alpha-\beta)}. \tag{3.12}$$

The required sample size is then

$$N = k\frac{\frac{\alpha\beta(2-\alpha-\beta)}{(\alpha+\beta)^3}}{\left\{\frac{r}{\Phi(\frac{1}{2}(1+s))}\right\}^2}, \tag{3.13}$$

where $\Phi(\cdot)$ is the standard normal CDF. k may be found by taking the smallest $k \in \mathbb{N}$ such that the first-order Markov chain model is preferred over a second-order Markov chain model for some model selection criteria. Often the optimal k is 1 and in other cases k is not required to be considered.

Pros:

- Gives bounds on error amount
- Provides amount of burn-in that should be discarded

Cons:

- Comparatively slow computation for a given number of samples taken
- Error bounds are on the CDF, not the quantile of interest
- Error of quantile is heavily tied to shape of posterior (how heavy tails are)
- Starting values may affect estimation adversely for larger error tolerances

3.4. Sequential Stopping Rules. Vats et al. [20] consider a vector Θ as defined in equation (2.1). Assume the existence of a functional CLT, which is established under the same conditions as (2.6), and a strongly consistent estimator for Θ and Σ. The procedure evaluates the estimated p^{th} volume of a $(1-\alpha)100\%$ confidence region and terminates once this value becomes suitably small. Under these assumptions the confidence region is

$$C_\alpha(n) = \{\Theta \in \mathbb{R}^p : n(\hat{\Theta} - \Theta)'\hat{\Sigma}^{-1}(\hat{\Theta} - \Theta) < T^2_{1-\alpha,p,df}\}, \tag{3.14}$$

where $T^2_{p,df}$ is a Hotelling's T^2 distribution with degrees of freedom p and df determined by the choice of estimator for $\hat{\Sigma}$. The accompanying volume is

$$Vol(C_\alpha(n)) = \frac{2\pi^{p/2}}{p\Gamma(p/2)} \left(\frac{T^2_{1-\alpha,p,df}}{n} \right)^{p/2} |\hat{\Sigma}|^{1/2}. \tag{3.15}$$

The stopping time is then defined as

$$T(\epsilon) = \inf\{n \in \mathbb{N} : (Vol(C_\alpha(n)))^{1/p} + a(n) \leq \epsilon K(X)\}, \tag{3.16}$$

where $a(n)$ is some function of $o(n)$ which prevents termination before some prespecified simulation effort, $\epsilon > 0$ and a metric of the estimation process $K(X)$ with $\hat{K}(X)$ a strongly consistent estimator. A common choice is to set $a(n) = \epsilon K(X) I(n \geq n^*) + \frac{1}{n}$ where n^* is a minimum sample size. $K(X)$ may be thought of as a weighting on the volume to scale what may be considered suitably small. If $K(X)$ is set to 1 a fixed volume sequential stopping rule is emitted. Other suggested choices for $K(X)$ are a relative magnitude $K(X) = ||\hat{\Theta}||$, the Euclidean norm of the estimate of Θ or a relative covariance $K(X) = |\hat{\Lambda}|^{1/2p}$ where $\hat{\Lambda}$ is the estimated posterior covariance based on X. These choices of $K(X)$ work from the idea that the tolerance for error should be based on how much variability is in the distribution of interest.

When a quantile ξ_q as defined in (2.7) is of interest, Doss et al. [5] develop a univariate procedure which is based on the length of the resulting confidence interval. Assume the existence of a functional CLT, which is established under the same conditions as (2.9), and strongly consistent estimators for ξ_q, $f_V(\xi_q)$, σ_q^2 and $k(X)$, a metric of the estimation process. Under these assumption the relative metric sequential stopping time is

$$t_\epsilon = \inf \left\{ n \in \mathbb{N} : Z_{1-\alpha/2} \frac{\hat{\sigma}_q}{\sqrt{n}\hat{f}_V(\hat{\xi}_q)} + a(n) \leq \epsilon k(X) \right\}, \tag{3.17}$$

where $a(n)$ and ϵ are defined as in the case of expectations. A choice of $k(X) = 1$ yields a fixed-width sequential stopping rule while other choices of $k(X)$ may be $|\hat{\xi}_q|$ or $\hat{\lambda}$ where λ is the posterior covariance.

As $\epsilon \to 0$ these stopping times yield asymptotic coverage probabilities of $1-\alpha$ as desired. In the rest of this report the batch means estimator for Σ and σ_q^2 will be used although any strongly consistent estimator is valid.

Pros:

- User controlled error tolerance
- Asymptotic coverage probabilities
- Diagnostic is cheap to compute when using batch means variance estimator
- Univariate or multivariate for expectations
- Handles expectations of functions and quantiles

Cons:

- Requires choice of ϵ with smaller choices yielding better results but more samples, results can become poor when ϵ taken to be too large
- Starting values can greatly influence how many samples must be taken
- Batch means variance estimator is slow to converge
- Checking ergodicity assumption is not simple. Generally must rely on previously studied methods/samplers

3.5. Heidelberger-Welch Diagnostic. Heidelberger and Welch develop a two stage procedure for the estimation of a steady state mean [12]. The first stage tests for stationarity of the process while the second stage compares the length of a confidence interval to a prespecified error tolerance weighted by the value of the mean estimate. This second stage is a special case of sequential stopping rules. Slightly different assumptions are required as the method is not designed for the MCMC context.

The stationarity test assumes a ϕ-mixing covariance stationary process and defines a quantity which converges in distribution to a Brownian bridge. This is then able to be tested using a statistic such as the Cramer-von Mises statistic to detect an initial transient state. This test is conducted using the entire sample. If it fails, then a beginning portion of the samples are ignored and the remaining portion is retested. The procedure is repeated until either a large amount of the samples are ignored in which case more samples must be generated or the test passes. Stationarity is not actually required in the MCMC setting as a LLN and CLT may exist without this assumption as discussed in [10]. Some may use this test as a method to check whether burn-in should be discarded, but this is unnecessary as discussed in Section 2.

The second stage of the process uses a relative magnitude sequential stopping rule with a spectral variance estimator. Conditions for which this procedure are asymptotically valid were later proved in [11] which included a FCLT and strongly consistent estimator for the limiting variance. The stopping time may be defined as,

$$\tau_\epsilon(n) = \inf\left\{ n \in \mathbb{N} : Z_{1-\alpha/2}\frac{\hat{\sigma}}{\sqrt{n}|\hat{\theta}|} < \epsilon \right\}. \tag{3.18}$$

Pros:
- Standard errors to quantify estimate
- Can scale error tolerance to scale of quantity of interest
- Can adjust for poor starting values

Cons:
- Requires choice of ϵ which can lead to poor results for large ϵ
- May discard large amounts of samples
- If quantity of interest is 0 causes division by ≈ 0
- Variance in quantity of interest should be related to magnitude of quantity of interest

4. Simulation Study. To measure the effectiveness of each diagnostic as a stopping rule two kinds of measures are considered. The first measure is of the quality of estimation in which the sum squared Monte Carlo error (SSMCE $= \sum(\hat{\theta}_j - \theta)^2$) based on replicated simulations is considered as well as empirical coverage probabilities for methods that generate confidence regions. Sum absolute Monte Carlo error was also considered, but in practice this did not provide much information beyond what SSMCE provided. A smaller value of SSMCE represents less error in the estimation and is preferred. The second measure is simulation effort for which the total time of the simulation and the average sample size at the stopping time are recorded. In general these are highly dependent, however there are some situations in which a method may take a longer simulation time but have less samples taken.

The implementation of MCMC methods in Dakota only uses Metropolis-Hastings Random Walk algorithms and as such this study will restrict itself to versions of this sampler. The Metropolis-Hastings Random Walk algorithm used may be described as follows.

For a target distribution π and a symmetric increment distribution f_ϵ, transition from X_i to X_{i+1} as follows:

1. Sample $\epsilon \sim f_\epsilon$
2. $X^* = X_i + \epsilon$
3. $a = \min\left\{1, \frac{\pi(X^*)}{\pi(X_i)}\right\}$
4. Let $X_{i+1} = \begin{cases} X^* & \text{w.p. } a \\ X_i & \text{w.p. } 1-a \end{cases}$
5. Increment i

The particular settings examined are

S_1: $\pi = N(2,2); \epsilon \sim N(0, 1/2)$

S_2: $\pi(x) = .3f_1(x) + .5f_2(x) + .2f_3(x)$, A mixture of three univariate normal distributions: $f_1 = N(1, 2.5), f_2 = N(5, 4), f_3 = N(11, 3); \epsilon \sim N(0, 9)$

S_3: $\pi = N_d(0, \Sigma = MM')$, where M is a $d \times d$ matrix with each entry $\overset{i.i.d.}{\sim} N(0,1); \epsilon \sim N_d\left(0, \frac{2.38^2}{d}\Sigma\right)$

where each distribution is parameterized by its variance or covariance matrix.

These are all fairly well behaved samplers and should not fail the diagnostics checks unless a poor increment distribution is chosen. As such this study is mostly informative in determining the effectiveness of the criteria as a stopping condition. The Geweke diagnostic and Gelman-Rubin diagnostic are not intrinsically stopping rules but will be considered for this analysis as they have been used in this manner.

In the following tables RL, Seq Q, Seq E, Gelman, Geweke and Heidel with stand for Raftery-Lewis diagnostic, sequential stopping rules for quantiles, sequential stopping rules for expectations, Gelman-Rubin diagnostic, Geweke diagnostic and Heidelberger-Welch diagnostic respectively.

4.1. Computing Time. To measure the computation time of each diagnostic independent of sampling time a data set of 10,000,000 univariate independent identically distributed random variables was generated. Each diagnostic was applied using the default settings in the packages "coda" [16] and "mcmcse" [6] from the statistical software R [17] to an increasing portion of the data sequentially in batches of 1,000,000 observations. The time is reported after 5,000,000 samples are included, denoted "Half", and after all 10,000,000 have been included, denoted "Full". The results are reported in Table 4.1.

	RL	Seq Q	Seq E	Gelman	Geweke	Heidel
Half	32.540	1.8212	0.37541	4.5266	16.687	39.629
Full	133.790	6.7323	1.45627	16.5546	68.193	169.713

Table 4.1: Simulation times in seconds for diagnostics applied to *i.i.d.* data.

The sequential stopping rules are incredibly quick to compute compared to the other methods. This is largely explained by the use of a batch means estimator compared to the spectral variance estimators used in the Geweke and Heidelberger-Welch approaches. The Heidelberger-Welch method has a two-stage approach in which the first stage should be responsible for the majority of the simulation time. In actual applications one would generally expect the time it takes to sample to be the largest simulation cost, but there is

potential for some circumstances under which another diagnostic may take less simulation effort with more samples taken.

4.2. Replicated Study. The following are a selection of some simulation results, with the full results being available upon request. Table 4.2 contains results of simulation setting S_1 when estimating the mean of the distribution. Sequential rules provide the smallest number of error but take the largest amount of samples, especially in the case of a "poor" starting value. A poor starting value includes when the initial sample is far from the mass of the distribution of interest. Gelman-Rubin is able to provide a reasonable estimate when the starting values become extreme enough, but these also add substantially to the simulation time. Additionally, the number of samples taken is actually m times the reported value as these are the number of samples per chain while there are m chains. In simple cases this diagnostic appears to have a tendency to terminate simulation very early and the error in the estimate suffers greatly. Heidelberger-Welch also takes a small number of samples leading to a larger error, but the error quantification it calculates provides close to correct coverage probabilities $(1-\alpha)$. The estimation for Geweke suffers when the starting value is poor. Additionally, Geweke terminates close to the minimum simulation effort of 10000 samples.

method	SSMCE	mean n	time	controls
Seq E	.9508519	45423	9.65	$\epsilon = .05, \alpha = .1$
Seq E	.7206755	64791	15.4	$\epsilon = .05, \alpha = .05$
Seq E	.6303106	203716	60.6	$\epsilon = .05, \alpha = .05$, x[1]=50
Gelman	24.51837	1441	.960	m=3, start=(20,-15,0)
Gelman	9.289586	1028	2.27	m=10
Gelman	4.632762	7569	5.67	m=3, start=(100,-1000,1000)
Gelman	0.645138	71021	82.4	m=3, start=(10000,-1000,1000)
Gelman	0.625250	71033	59.9	m=3, start=(10000,-1000,0)
Heidel	10.46866	4687	2.12	$\epsilon = .1, (\alpha = .05)$, coverage=.941
Heidel	9.933048	5926	3.21	$\epsilon = .1, (\alpha = .05)$, coverage=.928, x[1]=50
Heidel	2.60037	17653	10.4	$\epsilon = .05, (\alpha = .05)$, coverage=.95
Heidel	2.70864	19550	14.6	$\epsilon = .05, (\alpha = .05)$, coverage=.947, x[1]=50
Geweke	4.102102	14411	13.9	$\alpha = .25$
Geweke	185.7301	10000	3.26	$\alpha = .25$, x[1]=50
Geweke	4.330774	10205	2.19	$\alpha = .025$

Table 4.2: Results for 1000 simulated replications of Metropolis-Hasting Random Walk with Normal(2,2) target distribution. Time is reported in minutes.

Table 4.3 contains results from estimating quantiles using simulation setting S_2. Raftery-Lewis requires a fairly strict error tolerance to match sequential stopping rules with $\epsilon = .05$. Particularly of interest is that sequential stopping rules are actually faster in time although requiring more samples to achieve a smaller error. This is due to the expense of calculating Raftery-Lewis being much larger than sequential stopping rules. Also of interest is that the distribution being sampled is not symmetric and the effort to calculate quantiles with similar tail probabilities differs greatly. Raftery-Lewis performs better the closer the tails are to that of a Normal distribution.

Table 4.4 provides results for 100 replications of a 10-dimensional Normal distribution under simulation setting S_3. For each dimension of the distribution a 95% credible interval is estimated resulting in a $p = 20$ dimensional problem. The choice of a 95% credible interval is incredibly favorable to Raftery-Lewis as this requires quantiles heavily in the tails of the

method	SSMCE	mean n	time	controls
.1 quantile				
Seq Q	0.97707	86259.5	53.4	$\epsilon = .05, \alpha = .1$
RL	1.597042	53970	109.9	r=.005, s=.9
.9 quantile				
Seq Q	1.021776	320717.5	—	$\epsilon = .05, \alpha = .1$
RL	3.477389	91684	—	r=.005, s=.9
Median				
Seq Q	1.029267	193431.5	175.0251	$\epsilon = .05, \alpha = .1$
RL	2.850999	66096.5	188.9548	r = .01, s = .9
RL	13.95706	14293.5	10.27	r = .02, s = .9

Table 4.3: Results for 1000 simulated replications of Metropolis-Hasting Random Walk with target distribution a mixture of Normal distributions. Time is reported in minutes.

distribution resulting in less samples. Hence, the performance should decrease for shorter credible intervals. Additionally, the normality of the marginal posterior distributions should yield the smallest amount of error in the quantile estimation. Distributions with heavier tails should yield higher amounts of error. Sequential stopping rules achieve the desired coverage probability and the error amount may be considered acceptable based on the dimensionality of the problem. Expectations for the same distribution are also considered in Table 4.5 where the mean and variance of each dimension are estimated. Sequential stopping rules perform better than they did in the case of quantiles. Heidelberger-Welch runs into problems with its marginal approach and would crash on certain replicates causing the reporting of only the 38 replicates which occurred before the crash. This crash likely occurred due to memory limitations and with more careful planning should be avoidable. When it does not crash the marginal levels are restrictive enough to yield very good estimates, but this is arguably too restrictive in high dimensional problems. Additionally, Heidelberger-Welch falls short enough from its desired coverage probability that it may be considered an issue.

method	tr(SSMCE)	mean n	time(hours)	controls
RL	1.872033	1025000	12.54828	$r = .005, s = 1 - .05/p$
Seq Q	102.8902	3615000	37.73158	$C = .92, \epsilon = .1, \alpha = .1$

Table 4.4: 100 replications of a 10-dimensional Normal distribution. A .95 credible interval is estimated for each dimension leading to $p = 20$ quantiles being estimated. Time is reported in hours.

method	tr(MSMCE)	coverage	mean n	time/rep	reps	controls
Seq E	1.295256	.89	2275000	.2313	100	$\epsilon = .1, \alpha = .1$
Heidel	.2346839	.8421	3223684	.4857	38	$\epsilon = .1, \alpha = .1$

Table 4.5: Replications of a 10-dimensional Normal distribution. The mean and variance is estimated for each dimension leading to $p = 20$. Results are reported in a per replication scale as the Heidelberger-Welch simulation would crash multiple attempts. Time is reported in hours.

5. Conclusions. Questions to to consider when choosing a stopping criteria include:

- What is/are the quantity(ies) of interest?
- How many parameters are of interest?
- How much variability in the estimation is acceptable?
- Are extra samples expensive to compute?
- How important is speed of getting an estimate?
- How many computing resources are available?
- Is a reasonable starting value available?
- Is the quantity of interest close to 0?

Some of these questions are at odds with each other. A faster estimate is going to come with more variability in the estimation and may be at odds with interpretability of the error. If certainty in the estimate is of primary importance, then sequential stopping rules or Heidelberg-Welch should be the preferred methods with Heidelberg-Welch yielding larger amounts of error but being the faster method. If quantiles are of interest, then Raftery-Lewis is the better method for a quick estimate while sequential rules will yield more certainty and may be faster for higher levels of certainty when samples are cheap. If computing resources are plentiful (parallel computing available), then Gelman-Rubin may be considered. However, it may be best to combine Gelman-Rubin with another method, such as sequential stopping rules which may incorporate parallel chains into its procedure, for estimation purposes as Gelman-Rubin tends to terminate too early and does not provided an error estimate. If a starting value is known to be reasonable, then methods such as Gelman-Rubin or Geweke are not particularly useful and a method focused on estimation should be used as no burn-in will be required. If estimating a quantity close to 0, then Heidelberg-Welch should be avoided. When the dimension of the chain starts to become large, the conservative nature of marginal approaches starts to become problematic and sequential results should be used as they apply to the entire sampler. Raftery-Lewis may be preferred for error rates per sample when considering quantiles on the tail of a normal distribution but lose precision when the target distribution deviates from normality. In multivariate cases this allows it to sometimes greatly outperform sequential stopping rules in favorable conditions.

In general, sequential stopping rules for expectations yield useful results in a variety of settings and are the most flexible indicating that this would be a reasonable choice with a default setting with 90% confidence and $\epsilon = .05$ in low dimensions and $\epsilon = .1$ in high dimensions. If quantiles are of interest, Raftery-Lewis appears to be a reasonable choice if normality of the target distribution is reasonable. When the target is not normal sequential stopping rules may be more appropriate.

6. Acknowledgments. This study was funded by an NSF Mathematical Sciences Graduate Internship, conducted at Sandia National Laboratories, Albuquerque. Additional thanks go to Laura P. Swiler who provided large amounts of feedback during the reporting process and also helped provide context into the history and kinds of scenarios in which

Dakota is used.

REFERENCES

[1] B. M. Adams, M. S. Ebeida, M. S. Eldred, G. Geraci, J. D. Jakeman, K. A. Maupin, J. A. Monschke, J. A. Stephens, L. P. Swiler, D. M. Vigil, T. M. Wildey, W. J. Bohnhoff, K. R. Dalbey, J. P. Eddy, J. R. Frye, R. W. Hooper, K. T. Hu, P. D. Hough, M. Khalil, E. M. Ridgway, J. G. Winokur, and A. Rushdi, *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Paramter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.8 Reference Manual*, Sandia Technical Report SAND2014-5015, (July 2014. Updated May 2018 (Version 6.8)).

[2] P. Billingsley, *Probability and Measure*, Wiley, New York, third ed., 1995.

[3] S. P. Brooks and A. Gelman, *General methods for monitoring convergence of iterative simulations*, Journal of Computational and Graphical Statistics, 7 (1997), pp. 434–455.

[4] M. K. Cowles and B. P. Carlin, *Markov chain monte carlo convergence diagnostics: A comparative review*, JASA, 91 (1996), pp. 883–904.

[5] C. Doss, J. M. Flegal, G. L. Jones, and R. C. Neath, *Markov chain Monte Carlo estimation of quantiles*, Electronic Journal of Statistics, 8 (2014), pp. 2448–2478.

[6] J. M. Flegal, J. Hughes, D. Vats, and N. Dai, *mcmcse: Monte Carlo Standard Errors for MCMC*, Riverside, CA, Denver, CO, Coventry, UK, and Minneapolis, MN, 2017. R package version 1.3-2.

[7] J. M. Flegal and G. L. Jones, *Batch means and spectral variance estimators in Markov chain Monte Carlo*, The Annals of Statistics, 38 (2010), pp. 1034–1070.

[8] A. Gelman and D. B. Rubin, *Inference from iterative simulation using multiple sequences (with discussion)*, Statistical Science, 7 (1992), pp. 457–472.

[9] J. Geweke, *Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments (with discussion)*, in Bayesian Statistics 4. Proceedings of the Fourth Valencia International Meeting, J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, eds., Clarendon Press, 1992, pp. 169–188.

[10] C. J. Geyer, *Introduction to Markov chain Monte Carlo*, in Handbook of Markov Chain Monte Carlo, CRC, London, 2011.

[11] P. W. Glynn and W. Whitt, *The asymptotic validity of sequential stopping rules for stochastic simulations*, The Annals of Applied Probability, 2 (1992), pp. 180–198.

[12] P. Heidelberger and P. D. Welch, *Simulation run length control in the presence of an initial transient*, Operations Research, 31 (1983), pp. 1109–1144.

[13] G. L. Jones, *On the Markov chain central limit theorem*, Probability Surveys, 1 (2004), pp. 299–320.

[14] G. L. Jones, M. Haran, B. S. Caffo, and R. Neath, *Fixed-width output analysis for Markov chain Monte Carlo*, Journal of the American Statistical Association, 101 (2006), pp. 1537–1547.

[15] S. P. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, Springer-Verlag, London, 1993.

[16] M. Plummer, N. Best, K. Cowles, and K. Vines, *Coda: Convergence diagnosis and output analysis for mcmc*, R News, 6 (2006), pp. 7–11.

[17] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2017.

[18] A. E. Raftery and S. M. Lewis, *How many iterations in the Gibbs sampler?*, in Bayesian Statistics 4. Proceedings of the Fourth Valencia International Meeting, J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, eds., Clarendon Press, 1992, pp. 763–773.

[19] A. F. Seila, *Multivariate estimation in regenerative simulation*, Operations Research Letters, 1 (1982), pp. 153–156.

[20] D. Vats, J. M. Flegal, and G. L. Jones, *Multivariate Output Analysis for Markov chain Monte Carlo*, ArXiv e-prints, (2015).

[21] D. Vats, J. M. Flegal, and G. L. Jones, *Strong Consistency of Multivariate Spectral Variance Estimators in Markov Chain Monte Carlo*, Bernoulli, 24 (2018).

EXPLORING APPLICATIONS OF RANDOM WALKS ON SPIKING NEURAL ALGORITHMS

LEAH E. REEDER*, AARON J. HILL †, JAMES B. AIMONE ‡, AND WILLIAM M. SEVERA§

Abstract. Neuromorphic computing has many promises in the future of computing due to its energy efficient and scalable implementation. Here we extend a neural algorithm that is able to solve the diffusion equation PDE by implementing random walks on neuromorphic hardware. Additionally, we introduce four random walk applications that use this spiking neural algorithm. The four applications currently implemented are: generating a random walk to replicate an image, finding a path between two nodes, finding triangles in a graph, and partitioning a graph into two sections. We then made these four applications available to be implemented on software using a graphical user interface (GUI).

1. Introduction. There is growing interest in computing fields to utilize neural approaches; the importance of neuromorphic computing is growing, especially with its potential in beyond-Moore's Law techniques [8, 12]. Neuromorphic computing connects computing systems to neural systems in two ways: by using hardware to emulate the way the brain behaves and by using new architectures and paradigms that are brain inspired. The primary motivator for neuromorphic computing is to achieve high energy and volume efficiency, similar to human brains [4, 10].

In terms of power consumption and energy usage, nontraditional hardware is at a serious advantage in comparison to other traditional computing platforms [5, 9]. For random processes specifically, it can be seen that in some applications GPUs are more desirable than CPUs [17]. However, a recent paper showed that GPUs fall short on random simulations that are highly task-parallel rather than highly data-parallel [2]. Neuromorphic architectures can excell where GPUs fall short as they have similar parallelization capabilities but outperform energy-wise for random processes [6].

IBM's TrueNorth chip and the University of Manchester's SpiNNaker chip are two of the most popular neural-inspired hardwares. TrueNorth has a full custom application specific integrated circuit (ASIC) design that is highly optimized for a fixed spiking neuron model. Conversely, SpiNNaker is optimized for communication of the spike-based network and is flexible regarding the neuron model used. SpiNNaker, however, uses more energy than TrueNorth but still significantly less than traditional von Neumann architectures [10].

On the other side of neuromorphic computing are neural-inspired algorithms. Spiking neural algorithms for a specific random process, markov chain random walks, have been developed previously [1, 11]. It has been shown that the small neural circuits created from these algorithms are able to handle random walk simulations both efficiently and scalably. These algorithms were designed specifically to be used on neuromorphic architecture (both TrueNorth and SpiNNaker) in order to perform energy efficient simulations.

We present in this paper an extension to one of the algorithms previously developed as well as several important random walk applications utilizing the algorithm. Once these simulations are running on neuromorphic architecture, there will be significant power consumption advantages. Additionally, we will describe resulting visualizations of each application on an easy to use graphical interface that utilizes the spiking neural algorithm. Lastly, we will discuss hopes for further developments of the applications.

2. Background.

*Colorado School of Mines, lreeder@mines.edu
†Sandia National Laboratories, ajhill@sandia.gov
‡Sandia National Laboratories, jbaimon@sandia.gov
§Sandia National Laboratories, wmsever@sandia.gov

2.1. Introduction to Random Walks. Random walks are heavily studied stochastic processes and are well defined models of the motion a particle takes in a diffusion scheme. A 1-dimensional random walk without interactions is defined as a function $\tau : \mathbb{N} \to \mathbb{Z}$ where

$$\mathbb{P}(\tau(n+1) = x+1 | \tau(n) = x) := p \tag{2.1}$$

and

$$\mathbb{P}(\tau(n+1) = x-1 | \tau(n) = x) = 1 - p := q \tag{2.2}$$

Essentially, a particle at position x moves to the right $(x+1)$ with probability p and to the left $(x-1)$ with probability $1-p=q$. After n timesteps, the position of the particle is defined to be at some distance $m(t)$ from the origin. The 1-dimensional random walk generalizes easily to higher dimensions and ultimately to walks on general graphs [3, 15].

2.2. Diffusion Equation and Random Walks. The motivation for the random walk algorithms mentioned earlier was to solve partial differential equations (PDEs). As random walks model the motion of diffusion, they inherently solve the well known diffusion PDE. A derivation of the 1-dimensional diffusion equation as the limit of a random walk is outlined in Chapter 5 of [14]. We will summarize those steps here.

Assume a particle begins at the origin and has a step length of Δx and a timestep of Δt. Furthermore, assume that p is the probability that the particle moves Δx to the right and $q = 1 - p$ is the probability that it moves Δx to the left. Let the probability event that the particle is at position $x = m(\Delta x)$ with m steps to the right at time $t = n(\Delta t)$ after n time periods be denoted as:

$$p(m,n) = \text{probability of } r \text{ steps right} = \binom{n}{r} p^r q^{n-r}, \text{ where } r = \tfrac{1}{2}(n-m).$$

This is the binomial distribution, and can be approximated by a normal distribution using the Central Limit Theorem.

Let the function $u(x,t)$ represent the probability that the particle lies in an interval centered at x at time t with width $2\Delta x$. Then, we can denote $u(x,t)$ as the space- and time-scaled probability function, appropriately discretized as

$$u(x,t) = \frac{p(\frac{x}{\Delta x}, \frac{t}{\Delta t})}{2(\Delta x)}. \tag{2.3}$$

It can be shown that this simplifies nicely to the Gaussian distribution,

$$u(x,t) = \frac{e^{-\frac{x^2}{4Dt}}}{\sqrt{4\pi D t}} \tag{2.4}$$

where D is the diffusion coefficient and is defined as

$$D = \frac{\Delta x^2}{2(\Delta t)}. \tag{2.5}$$

The function $u(x,t)$ satisfies the following PDE (the diffusion equation)

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2}. \tag{2.6}$$

Thus, the diffusion equation can be derived from a 1-dimensional random walk. As stated previously, this case is easily generalizable to higher dimensions, which is critical for generating random walks on arbitrary graphs.

2.3. Introduction to Spiking Algorithm. The spiking neural algorithm discussed here was first presented in [11]. This paper presents two spiking algorithms for random walks, the particle method and the density method. The goal of these algorithms is to solve the diffusion equation PDE stochastically using random walks without interaction on neuromporhic architecture. We will focus on the density method only.

In the density method, rather than tracking each individual walker's position at any given point in time, we can instead keep track of the nodes on the graph and count how many walkers are at each node at any given point in time. When we are running the spiking simulation, we embed a spiking circuit that is located at each node in the graph. An outline of a frame of the random walk simulation utilizing the density method is described below.

1. An injected current starts initial walkers at a given node in graph by sending signals to walker generator neurons and walker counter neurons
2. Walkers are distributed throughout the circuit at that node through excitatory signals from the walker generator neurons
3. When a signal is received by output gate neurons, their potentials are modified
4. If the output gate neuron's thresholds are met, the output gate neurons determine whether or not to spike based off of a certain probability (discussed in next section)
5. If the neurons spike, a signal is sent from the output gate neurons in the current circuit to the determined walker generator neurons and walker counter neurons at the neighboring node's circuit

This process is followed generally in each timestep until there are walkers spread out over all circuits throughout the graph. Originally, the code only supported up to 2 (1-dimensional graphs) or 4 (2-dimensional graphs) output neurons per circuit, which corresponds to only 2 or 4 neighbors on a graph. However, this has been extended to support arbitary graphs with any number of neighbors. This extension is outlined in the next section.

The density method is implemented on TrueNorth. It currently supports only 4 neighbors, but we are working on extending it to support arbitrary graphs. This method implemented on TrueNorth has been shown to have significant power advantages compared to random walk codes on traditional architectures [1].

3. Extending the Current Algorithm.

In order to make the code generalizable to more than just 1D and 2D graphs, more neighbors needed to be supported. This is useful for applications such as social network graphs, where each node has significantly more connections than just 4.

We were able to develop a stochastic algorithm that supports any number of neighbors. The direction that a neuron is going to take is determined stochastically through a probability tree. Once a certain type of neuron is signalled to the root node in the tree, walkers are subsequently propagated down through the tree. This probability tree works by always receiving a spike through a walker generator neuron. The generator neuron then has connections with stochastic neurons that randomly spike. If a stochastic neuron spikes, it sends excitatory signals to the left half of the tree and inhibitory signals to the right half. These types of signals propagate throughout the tree until they reach output signals at the bottom of the tree.

Each stochastic neuron in the probability tree has two outputs (excitatory and inhibitory), meaning the algorithm currently best supports base 2 number of output directions. That is, graphs with base 2 number of neighbors (2, 4, 8, etc.) are optimal. If the graph has a differing number of neighbors, there will always be more output neurons than needed.

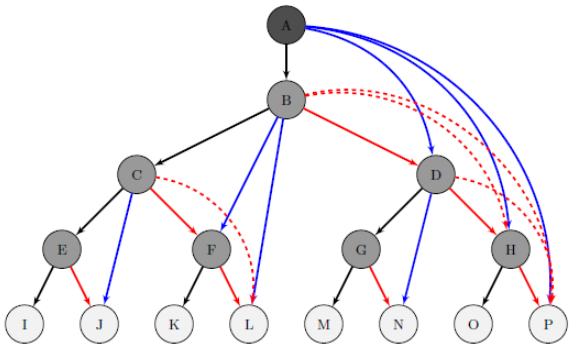


Fig. 3.1: A probability tree with 8 output gate neurons (8 directions). The *walker generator neuron* is depicted in dark grey and labeled as A, the *random gate neurons* are depicted in grey and are labeled as B-H, and the *output gate neurons* are depicted in light grey and are labeled as I-P. The connections are colored as follows: excitatory connections with delay 1 are black (blue with higher delays), inhibitory connections with delay 1 are red (dashed red with higher delays)

This has potential memory cost limitations as there often will be non-optimal numbers of neighbors, but has relatively low impact on the rest of the simulation.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Direction
1	1	1	0	1	0	-	0	1	0	-	0	-	-	-	0	Direction 1
1	1	1	0	*	0	-	0	-	1	-	0	-	-	-	0	Direction 2
1	1	*	0	-	1	-	0	-	-	1	0	-	-	-	0	Direction 3
1	1	*	0	-	*	-	0	-	-	-	1	-	-	-	0	Direction 4
1	0	-	1	-	-	1	0	-	-	-	-	1	0	-	0	Direction 5
1	0	-	1	-	-	*	0	-	-	-	-	-	1	-	0	Direction 6
1	0	-	*	-	-	-	1	-	-	-	-	-	-	1	0	Direction 7
1	0	-	*	-	-	-	*	-	-	-	-	-	-	-	1	Direction 8

Table 3.1: A schematic of the probability tree with 8 directions. The walker generator neuron is labeled as A, the random gates are labeled as B-H and the output gate neurons are I-P. Each output gate neuron corresponds to a different direction. If a neuron recieves an excitatory signal and spikes, it is denoted by a '1'. If a neuron recieves an inhibitory signal and does not spike, it is denoted by a '0'. If a neuron recieves an excitatory signal but does not spike, it is denoted by a '*'. Nodes that do not recieve a signal or spike are denoted by a '-'.

An example of a probability tree with 8 output gate neurons can be seen in Figure 3.1 and a supplementary schematic can be seen in Table 3.1. This is especially useful to see what combination of random neurons are needed in order for each output neuron to spike.

4. Applications of Random Walks. After running a random walk simulation, the resulting walk can be used in applications such as social network analysis and graph theory. Data analytics is difficult on large graphs. It can be shown that some of the more traditional graph algorithms do not scale well on larger graphs due to being sequential in nature [7]. We

expect that it could be beneficial to analyze the results of the random walk simulation rather than the actual graph. By generating these applicatons on a random walk, we can instead implement these algorithms in a nontraditional approach. Here we present four applications that we have implemented using the spiking neural algorithm introduced earlier, instead of using the graph dataset.

4.1. Using an image as underlying graph structure. The first application we implemented was using an image as the mesh. Instead of running the simulation on a graph with arbitrary probabilities, the probabilities are determined by the color of each pixel in the image. A noise constant can be used that serves to decrease the variation in each probability without changing the underlying structure.

One application that this method is particularly useful for is defining energy landscapes. For example, if the different colors in the image correspond to an energy gradient, then the directional probabilities assigned to each pixel in the graph would define the resulting energy landscape. When the random walk is generated, there is a higher number of walkers at the ideal energy levels. The pseudocode is listed in Algorithm 8 and an example of the output is depicted in Figure 5.4.

Algorithm 8 Image to Walk

```
read in image file
specify initial walker starting points
specify noise constant
convert file into matrix (each pixel in image is converted to a number based on color)
for each entry in matrix do
    determine 4 neighboring entry locations (up, down, left, right)
    determine 4 neighboring entry values (number corresponding to color)
    calculate probabilities for all 4 neighboring entries
    the probability is determined by the value of the neighboring entries
    (the darker the pixel, the more likely the walker will go to that pixel)
end for
generate random walk
```

4.2. Path finding on network graphs. The second application that is implemented is the classic path finding algorithm. Given two specified nodes, this algorithm can determine if there is a path between them using the random walk simulation data.

Finding paths in graphs is commonly used in road networks to find directions from point A to point B. It can also be used for flight path connections between different airports. Overall, this application is best used on arbitrary network graphs. The pseudocode is listed in Algorithm 9 and an example of the output is shown in Figure 5.6.

4.3. Triangle Finding on Network Graphs. The third application that is implemented is an algorithm that finds triangles on an arbitrary graph. Given a specific dataset, a random walk is generated for each node in the graph. Using the data from the random walk simulation, it can determine whether or not there was a triangle at each node.

Triangle finding can be used to find close-knit communities with obvious links between them. Many social network sites can use found triangles to suggest ‘friends’ or ‘connections’. Many traditional triangle finding or counting algorithms utilize the square of adjacency or incidence matrices [16]. With arbitrary graphs especially, these matrices can get very large and costly quickly. Utilizing random walks to find triangles can help in this regard as this nontraditional triangle finding algorithm does not deal with adjacency or incidence matrices.

Algorithm 9 Path Finding

```
read in network data
initialize graph from data
specify source and target nodes
for each node in graph do
    determine the number of neighbors that node has
    the probability is proportional to the number of neighbors that node has (assuming all edges have equal weights)
    assign probability to each neighboring node
end for
generate random walk with initial walkers at source node
for all timesteps in simulation do
    if neurons at target node spiked at time then
        there is a path from source to target
    else
        there is not a path from source to target
    end if
end for
```

The pseudocode is listed in Algorithm 10 and an example of the output is shown in Figure 5.9.

Algorithm 10 Triangle Finding

```
read in network data
initialize graph from data
for each node in graph do
    for each node2 in graph do
        determine the number of neighbors that node2 has
        the probability is proportional to the number of neighbors that node2 has (assuming all edges have equal weights)
        assign probability to each neighbor of node2
    end for
    generate random walk with initial walkers at node
    if neurons at node spiked at time=0 and time=3 then
        node is in a triangle
    end if
    to find other nodes in same triangle, look through spike log to see which neurons sent and recieved the spikes starting at time=0 through time=3
end for
```

4.4. Graph Partitioning on Network Graphs. The fourth application that is implemented is an algorithm that partitions a graph into two parts. Given a specific dataset, a random walk is generated that starts at the first node in the set. Using the data from the random walk simulation, the time at which neurons at each node spiked can be determined. If it took walkers a 'long' time (above a certain threshold) to get to a certain section of the graph, then the nodes in that section become their own partition.

A common application of graph partitioning is image segmentation [13]. To segment an image, often there are clear sections that correspond to different parts of the image. Once these segmentations are partitioned, the image (or graph) is segmented. This is useful to identify certain parts of the image as being different from others. The pseudocode is listed in Algorithm 11 and an example of the output is shown in Figure 5.11.

Algorithm 11 Graph Partitioning

```
read in network data
initialize graph from data
specify threshold
for each node in graph do
    determine the number of neighbors that node has
    the probability is proportional to the number of neighbors that node has (assuming all edges have equal weights)
    assign probabilities to each neighboring node
end for
generate random walk with initial walkers at initial node
for all neurons at each node do
    determine the time at which each neuron first spiked (if at all)
    if time at which they spiked is greater than the threshold then
        node is put into second partition
    end if
end for
```

5. A Graphical User Interface for Applications of Random Walks. Here we present a Graphical User Interface (GUI) for users to be able to use a plethora of applications of random walks on our spiking neural framework. This is a nice, clear way to see the useful applications of random walks.

The visualizations shown in three of the GUI applications (Path Finding, Triangle Finding, and Graph Partitioning) depict the user-specified graphs. The nodes are sized and colored based on howmany connections they have (unless the color is otherwise specified on the legend).

The main menu, with the four random walk applications can be seen below.

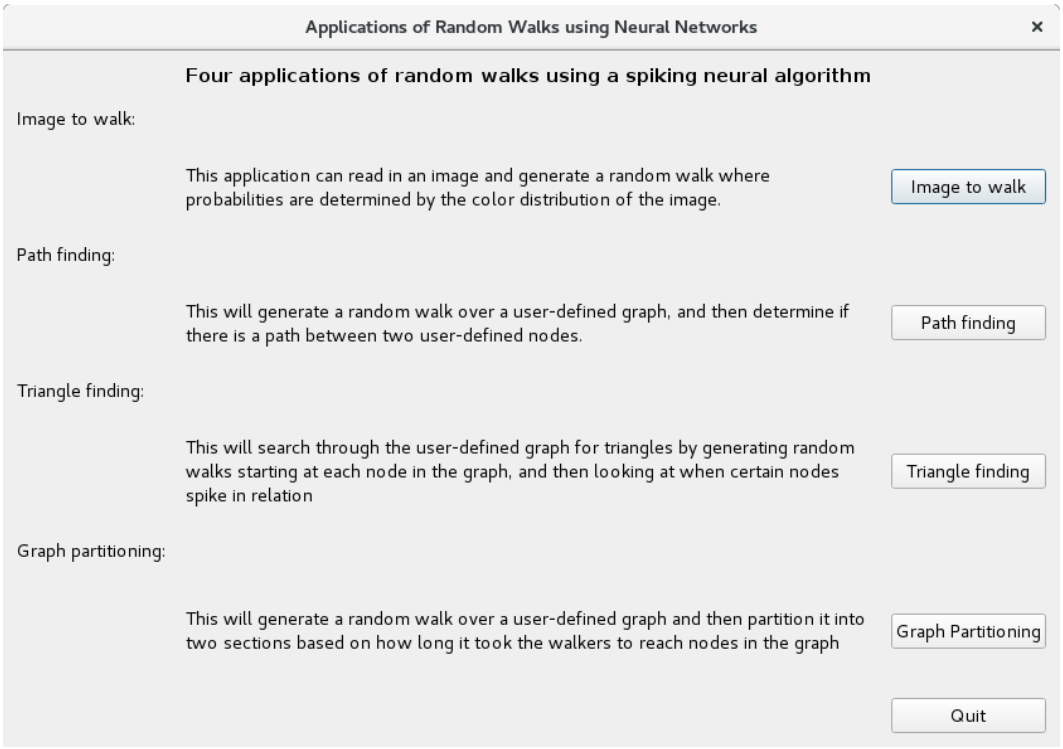


Fig. 5.1: Main menu: Gives four application options, with summaries of each

5.1. Image to Walk. The first application, Image to Walk can be seen below. The user is asked to upload the image in array form. This is because there are many different ways to convert an image into an array with various condensing techniques. By letting the user convert the image themselves, they can control what kind of condensing they will get. A pop-up message indicates when the simulation is complete that specifies the name of the resulting file to view the results. This file is stored in the same directory as the GUI files.

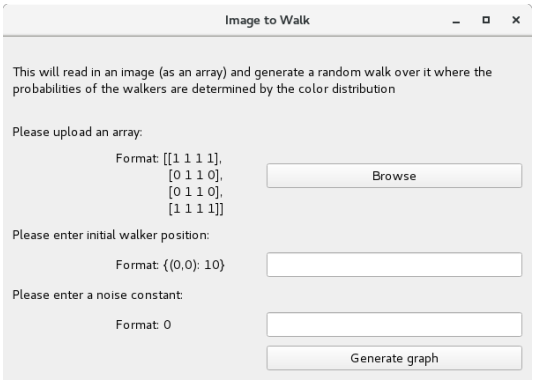


Fig. 5.2: Image to walk menu: allows user to specify array, initial walker locations/size, noise constant

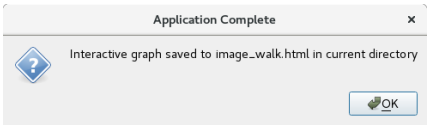


Fig. 5.3: Pop-up message that indicates when simulation is complete/ where output file is stored

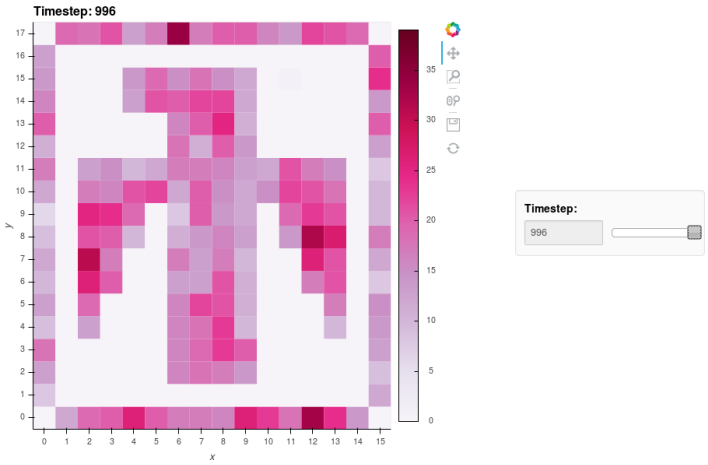


Fig. 5.4: Resulting image from Image to Walk application at last timestep

5.2. Path Finding. The second application, Path Finding, can be seen below. The user is asked to choose the file that the graph data set is in, specify the source node, and specify the target node. Once the simulation is done running, if there is a path from the source node to the target node, a visualization of the data set showing the path from the source to the target node pops up onto the screen.

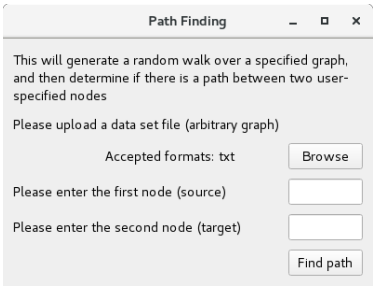


Fig. 5.5: Path finding menu: allows user to choose dataset file and specify source/target nodes

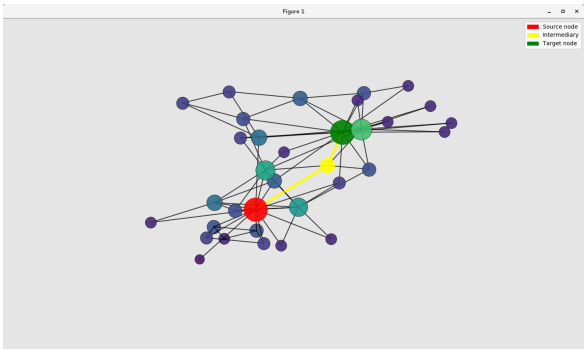


Fig. 5.6: An example of a visualization of a path. This dataset is Zachary's Karate Club [18]

5.3. Triangle Finding. The third application, Triangle Finding, can be seen below. The user is asked to choose the file that the graph data set is in. Once the simulation is complete, a visualization of the data set and all of the triangles in it pops up onto the screen. Note that since the triangle finding algorithm is done on the random walk, it often does not find all triangles in the dataset, just some. This is because walkers will not go in all directions every time the simulation is run. A limitation with this application is that sometimes the nodes are in multiple triangles and the visualization does not support that case. In this case, that node only shows one of the triangles that it is in, not both. As this application generates a random walk for each node in the graph, it takes a while to run. There is a popup message before the application starts running to warn the user that it will take a while.

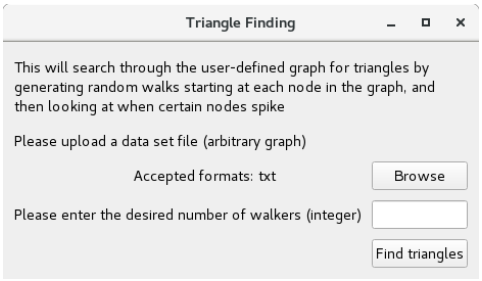


Fig. 5.7: Triangle finding menu: allows user to choose dataset file

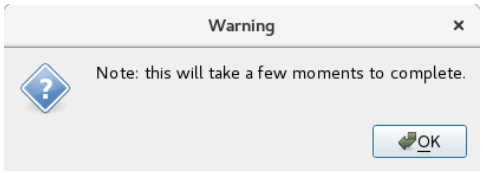


Fig. 5.8: Pop up message before application starts to run, warning user of long runtime

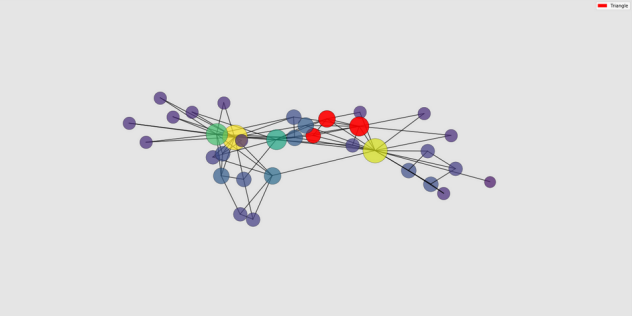


Fig. 5.9: An example of a visualization of one triangle in a dataset. This dataset is Zachary's Karate Club [18]

5.4. Graph Partitioning. The last application, Graph Partitioning, can be seen below. The user is asked to choose the file that the graph dataset is in. Once the simulation is complete, a visualization of the two partitions pops up onto the screen. As the partition algorithm runs on the random walk output on an undirected and unweighted graph, the graph gets partitioned differently every time. A limitation of this is that occasionally it would be better for the graph to be partitioned into three or four graphs, and the algorithm only splits it into two partitions.

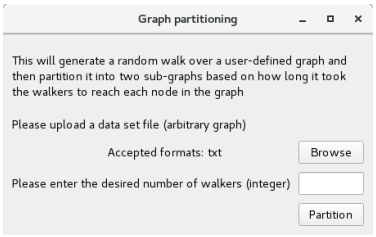


Fig. 5.10: Graph partitioning menu: allows user to choose data set file

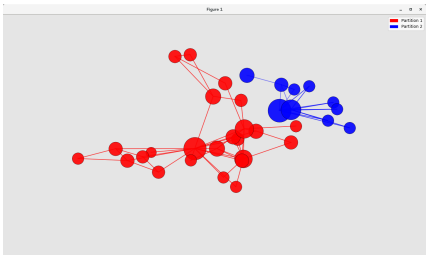


Fig. 5.11: An example of a visualization of a partitioned dataset. This dataset is Zachary's Karate Club [18]

6. Future Work. In the future, it would be advantageous to generate even more random walk applications to use with the spiking neural algorithm. One specific application that would be useful is image segmentation. Another important next step is getting all of the applications up and running on TrueNorth, a neuromorphic chip. Although the applications work on the software by themselves, we can really see vast performance advantages on neuromorphic hardware [6]. If we could get these applications running on the hardware, we could potentially have the user specify on the GUI whether they would like to run the random walk application on the spiking software, or the spiking hardware available (TrueNorth, SpiNNaker, Loihi, etc). This would be particularly useful to compare the performance of the different platforms.

7. Conclusion. With these applications implemented on a neural algorithm, they have the potential to be more energy-efficient and have less power-consumption compared to their traditional counterparts. Once implemented on neural hardware, there will be significant

performance advantages, even without being seriously optimized. It is important to consider these non-neural applications that utilize neural frameworks in order to promote more use of nontraditional architectures.

8. Acknowledgement. This research was funded by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract de-na0003525.

REFERENCES

[1] J. B. Aimone, A. Hill, R. Lehoucq, O. Parekh, and W. Severa, *Neuromorphic hardware implementation of spiking algorithms for markov random walks*, in Proceedings of the 2018 International Conference on Neuromorphic Systems (ICONS), 2018. To appear.

[2] R. M. Bergmann, K. L. Rowland, N. Radnović, R. N. Slaybaugh, and J. L. Vujić, *Performance and accuracy of criticality calculations performed using warp – a framework for continuous energy monte carlo neutron transport in general 3d geometries on gpus*, Annals of Nuclear Energy, 103 (2017), pp. 334 – 349.

[3] P. C. Bressloff, *Stochastic processes in cell biology*, vol. 41, Springer, 2014.

[4] A. Calimera, E. Macii, and M. Poncino, *The human brain project and neuromorphic computing*, Functional neurology, 28 (2013), p. 191.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, *A performance study of general-purpose applications on graphics processors using cuda*, Journal of parallel and distributed computing, 68 (2008), pp. 1370–1380.

[6] A. J. Hill, J. W. Donaldson, F. H. Rothganger, C. M. Vineyard, D. R. Follett, P. L. Follett, M. R. Smith, S. J. Verzi, W. Severa, F. Wang, et al., *A spike-timing neuromorphic architecture*, in Rebooting Computing (ICRC), 2017 IEEE International Conference on, IEEE, 2017, pp. 1–8.

[7] J. A. Miller, L. Ramaswamy, K. J. Kochut, and A. Fard, *Research directions for big data graph analytics*, in 2015 IEEE International Congress on Big Data, June 2015, pp. 785–794.

[8] D. Monroe, *Neuromorphic computing gets ready for the (really) big time*, Commun. ACM, 57 (2014), pp. 13–15.

[9] D. Ristè, M. P. da Silva, C. A. Ryan, A. W. Cross, A. D. Córcoles, J. A. Smolin, J. M. Gambetta, J. M. Chow, and B. R. Johnson, *Demonstration of quantum advantage in machine learning*, npj Quantum Information, 3 (2017), p. 16.

[10] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, *A survey of neuromorphic computing and neural networks in hardware*, arXiv preprint arXiv:1705.06963, (2017).

[11] W. Severa, R. Lehoucq, O. Parekh, and J. B. Aimone, *Spiking neural algorithms for markov process random walk*, arXiv preprint arXiv:1805.00509, (2018).

[12] W. Severa, O. Parekh, K. D. Carlson, C. D. James, and J. B. Aimone, *Spiking network algorithms for scientific computing*, in Rebooting Computing (ICRC), IEEE International Conference on, IEEE, 2016, pp. 1–8.

[13] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888–905.

[14] R. W. Shonkwiler and F. Mendivil, *Explorations in Monte Carlo Methods*, Springer Science & Business Media, 2009.

[15] L. Sjogren, *Chapter 2 : Random walks [pdf]*.

[16] C. E. Tsourakakis, *Counting triangles in real-world networks using projections*, Knowledge and Information Systems, 26 (2011), pp. 501–520.

[17] D. Van Antwerpen, *Improving simd efficiency for parallel monte carlo light transport on the gpu*, in Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, ACM, 2011, pp. 41–50.

[18] W. Zachary, *An information flow model for conflict and fission in small groups*, Journal of anthropological research, 33 (1976).

PARTIALLY STRUCTURED AGGREGATION FOR MULTIGRID

PETER OHM*, LUC BERGER-VERGIAT†, AND RAY TUMINARO‡

Abstract. Multigrid methods have been developed for both structured and unstructured grids [3]. Unstructured meshes are often favored in simulations as they more easily represent complex geometries. However, mesh structure can often be exploited by the solver for better performance. In many circumstances an may only require complex unstructured meshes in small sub-regions of the overall domain. For these applications, we have developed an aggregation based multigrid method that preserves local structure, allowing the local use of structure specific methods.

1. Introduction. This paper considers the implementation of multigrid as a preconditioner for applications where the underlying mesh contains both structured and unstructured regions. In particular, we focus on the development of coarsening techniques for these partially structured grids. Some examples of these partially structured grids are shown in Figure 1.1, and such grids arise naturally in many applications. Despite having regions of structure, they must overall be treated as an unstructured mesh.

Multigrid methods have been developed for both structured and unstructured grids [3]. Unstructured meshes are often favored in simulations as they more easily represent complex geometries. However, mesh structure can often be exploited by the solver for better performance. This includes as structure specific smoothing, such as line smoothers, or structure specific coarsening such as geometric multigrid or black box algebraic multigrid [4]. An advantage of using structured coarsening is the preservation of the mesh structure on each level of the multigrid hierarchy. Then having a structured mesh on each level makes it easy to apply structured techniques, such as line smoothing, on each level of the hierarchy. These advantages are demonstrated in [7].

Here, we introduce a hybird aggregation scheme for semi-structured meshes that contain both regions of structured mesh and regions of unstructured mesh. The intent is to use structure-based algorithms to build the multilevel interpolation operator in the structured regions and to use unstructured or algebraic algorithms in the unstructured regions. The result is a multigrid method that preserves the partially structured nature of these meshes.

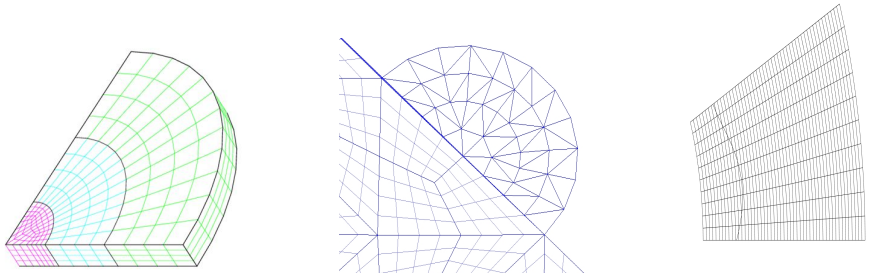


Fig. 1.1: Radial tri-section mesh (left), unstructured region attached to an HHG mesh (middle), interface with cut element mesh (right).

The rest of the paper is outlined as follows. Section 2 discusses a hybrid aggregation for partially structured meshes. Section 3 covers the implementation of this hybrid aggregation

*Tufts University, peter.ohm@tufts.edu
†Sandia National Laboratories, lberge@sandia.gov
‡Sandia National Laboratories, rstumin@sandia.gov

in MueLu [5, 9], a multilevel solver package for Trilinos. Finally concluding remarks are given in Section 5.

2. Hybrid Aggregation. The driving idea behind hybrid aggregation is to aggregate regionally based on the local structure of the region. Throughout this paper we use the term "aggregate" in a loose sense to mean the collection of fine degrees of freedom that contribute to a coarse node in a multigrid framework. This allows us to discuss both geometric and algebraic multigrid methods in the same setting.

Hybrid aggregation is based on a regional approach that heavily borrows from multigrid solvers for hierarchical hybrid grids (HHGs) [1,2]. HHGs are formed by the regular refinement of an initial coarse grid, resulting in a grid hierarchy containing regions of structured mesh, even if the initial coarse mesh is unstructured. The multigrid interpolation operators are then constructed regionally on the HHG hierarchy, allowing for structured multigrid methods to be used locally.

Given a fine mesh that can be decomposed into regions of structured mesh and regions of unstructured mesh, such as in Figure 2.1, one can regionally construct the multigrid interpolation operators based on the structure of the local region. Specifically, if the region is structured one uses a structure-based aggregation method, and if the region is unstructured then a standard aggregation method is used. In Figure 2.1, the left half of the mesh forms a structured region, the right half of the mesh forms a unstructured region, and the two regions share an interface in the middle of the mesh. For the most part, the multigrid interpolation operator for each region can be constructed independently of neighboring regions, however, special care may be needed on the interfaces between adjacent regions in order to ensure that the resulting coarse interface between adjacent regions remains a conforming mesh on the coarse level.

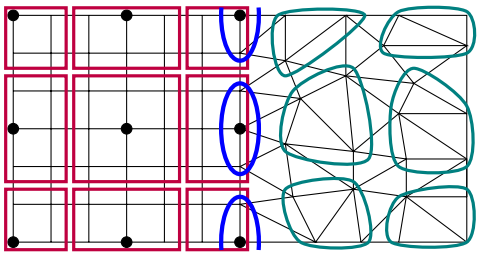


Fig. 2.1: Example of hybrid aggregation on a partially structured mesh. The left half of the mesh is structured, while the right half of the mesh is unstructured. In the structured region the points show the coarse nodes and the thick rectangles represent the structured aggregates. The more organic shapes represent the aggregates on the unstructured region. The interface between structured and unstructured regions can require special treatment. In the image above, the degrees of freedom on the interface are duplicated between the structured and unstructured regions, so special treatment is needed. The ellipses along the interface are special aggregates on the unstructured region, ensuring a conforming coarse mesh.

2.1. Shared Interfaces. A concern with the hybrid aggregation method is how to handle interfaces between regions using a structured multigrid algorithm and regions using an unstructured algorithm. This is primarily a concern when one wants or needs degrees of freedom to be duplicated or shared on or across regional interfaces. Differing relative coarsening rates between regions or disagreement in coarse node selection on an interface between regions may result in a coarse level mesh that is no longer conforming.

The primary way to ensure alignment of the interface nodes on the coarse meshes is to force the aggregation of the interface nodes to match on the interface between adjacent regions. For example, if nodes i and j are nodes on the shared interface between region A and region B, and are part of the same aggregate on region A, then i and j will also be part of the same aggregate on region B. Forced interface agreement is demonstrated in Figure 2.1. The degrees of freedom, element vertices, on the shared interface are duplicated between the structured and unstructured region. The ellipses show the required interface aggregates on the unstructured region to produce a conforming coarse level mesh. Since it is preferable to preserve the structure on the structured regions, the structured aggregates on the interface are used as guides to influence which interface nodes should be aggregated together on the unstructured region. Hence the logical structure on the structured region is preserved and the coarse mesh will be conforming across the interface.

3. MueLu Implementation. A hybrid aggregation scheme is implemented in the MueLu package for Trilinos, which serves as a multigrid solver library. A general outline for the setup of a MueLu operation is shown by the diagram in Figure 3.1. Setup for MueLu begins with the creation of the smoother, or relaxation method, for the current level of the multilevel hierarchy, followed by the construction of the transfer operators. The transfer operators are then used by the RAP factory to project the current solution to the coarse level. Finally there is rebalancing to calculate load balancing for parallel runs.

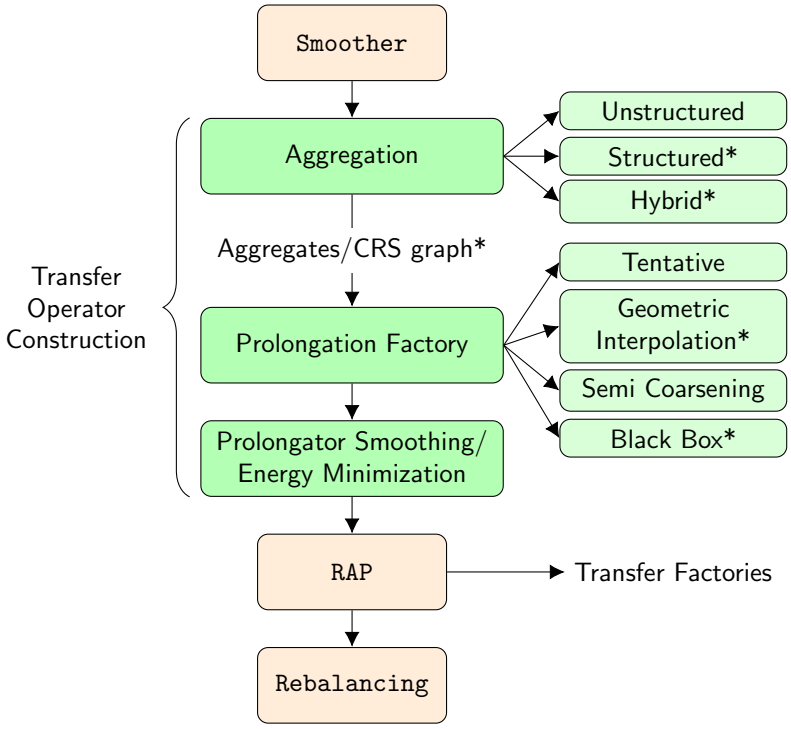


Fig. 3.1: Diagram of MueLu setup structure. (*) indicates aspects that are touched on in this paper.

Here, we focus on the transfer operator construction, which is comprised of three primary stages. The first stage is the aggregation factory, where the fine level is aggregated

into the coarse degrees of freedom. This is followed by the transfer factory, where the prolongation and restriction operators are constructed from the aggregates. The final step is optional enhancement of the prolongation and restriction operators, such as through energy minimization [6,8] or smoothing for smoothed aggregation [10,11].

MueLu has existing aggregation factories for both matrix-based aggregation, such as smoothed aggregation AMG, as well as mesh-structure based aggregation, which mimics a geometric multigrid approach. As discussed in section 2, the goal of hybrid aggregation is to aggregate on a regional scale, using the most ideal aggregation method for each region. To accomplish this, we implement a hybrid aggregation factory within the aggregation factory to perform region-based aggregation.

The hybrid aggregation factory takes a user provided region type and performs the appropriate aggregation for the region. The aggregation types currently supported by the hybrid aggregation factory are matrix-based uncoupled aggregation and structured aggregation. The setup is similar to the corresponding aggregation factory and the aggregation is preformed in ordered phases. The aggregation phases for hybrid aggregation are the same phases used by the structured aggregation factory or the uncoupled aggregation factory, and the hybrid aggregation factory arranges these existing aggregation phases to be compatible with the region type. The idea behind this construction is that in cases where structured information is needed, the structured aggregation phases are run first and the information from that phase is extracted and used in later aggregation phases, such as interface aggregation.

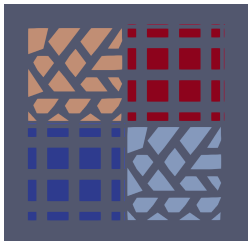


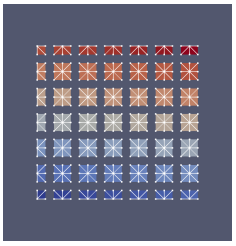
Fig. 3.2: A visualization of the hybrid aggregation. The bottom left and top right regions use structured aggregation while the top left and bottom right regions use uncoupled unstructured aggregation.

An example illustrating such an approach is seen in Figure 3.2. A uniform square domain is divided into four regions, where the top left and bottom right reigons are flagged as unstructured mesh regions, for which the hybrid aggregation scheme uses uncoupled unsmoothed aggregation. The top right and bottom left regions are treated as structured, and use an uncoupled geometric piece-wise constant interpolation scheme to construct aggregates. Since we are using uncoupled and unsmoothed aggregation with no node duplication on the shared interface, we do not have to worry about coarse node agreement on the interfaces between regions.

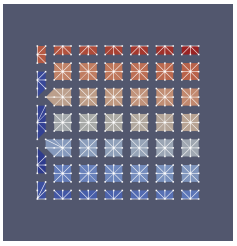
As mentioned in section 2.1, interfaces that are shared between regions can be problematic. As a step to alleviate these problems, we have implemented an interface aggregation factory. The interface aggregation factory is used to aggregate nodes on a shared interface so that the coarse level of adjacent regions still match on the shared interface. Since the coarse level from the structured aggregation factory is determined by the structure of the mesh, and is in turn structured, we do no want to change the structured aggregates. Thus, the interface aggregation factory is intended to be used as an aggregation phase during uncoupled aggregation. The interface aggregation factory takes a list that specifies which nodes are on

the interface and a list that says which of the interface nodes should be aggregated together. The lists agree with the aggregates from the neighboring structured region that lie on the interface. This interface aggregation phase is preformed before the aggregation phases for uncoupled aggregation, ensuring that the aggregates on the interface match the aggregates of the neighboring region.

A comparison between using interface aggregation to help ensure agreement and doing fully uncoupled aggregation can be seen in Figures 3.3a and 3.3b. Although the resulting aggregates using interface aggregation are not as uniform, they preserve properties that can be more important, such as agreement of coarse nodes on shared interfaces.



(a) No Interface Aggregation



(b) Interface Aggregation

Fig. 3.3: Comparison between using interface aggregation and not using interface aggregation. In Figure 3.3b five root nodes are specified on leftmost edge.

Further ideas that have not yet been implemented are other interface correction methods. It may be necessary or desirable to implement some post-processing of the transfer operators so that the transfer operators on the shared interface agree with each other. Currently, the interface aggregation only results in agreement in the sparsity pattern of the transfer operators on the shared interface. Different approaches can be used to reach interface operator agreement, such as averaging the transfer operators on the shared interface or simply choosing to use the interface transfer operator from one region on both.

The current implementation is limited to one region per rank, meaning only one mesh type per rank. For a given rank, the mesh must be treated as either fully structured or fully unstructured. Future work seeks to expand this to allow multiple regions and mesh types per rank, meaning a single rank could contain both unstructured and structured regions.

4. Numerical Results. For a test problem we use hybrid aggregation multigrid to precondition a GMRES solve for a 3D poisson problem on the unit cube with a uniform hexahedral mesh and 10^6 degrees of freedom. The domain was divided into two regions. Three different transfer operators were considered. First, an entirely structured operator, meaning that structured aggregation was used in both regions. Next, a hybrid structured-unstructured operator, meaning that one region used structured aggregation while the other region used an unstructured aggregation algorithm. Finally, an entirely unstructured operator, using unstructured aggregation in both regions. For all the tests the structured aggregation algorithm uses uncoupled piece-wise constant geometric interpolation and the unstructured aggregation uses uncoupled piece-wise constant aggregation. The smoother is a symmetric Gauss-Seidel method. The exact problem used was `HybridIntrepidPoisson3D_Pamgen_Tpetra` from the scaling examples in the `trilinoscoupleings` package. The parameter XML file used was `muelu_hybrid_aggregation_3D.xml`, with the mesh size modified. The number of iterations

	Structured-Structured	Structured-Unstructured	Unstructured-Unstructured
Iterations	28	29	30
Avg. convergence rate	0.519	0.536	0.545

Table 4.1: Iteration counts for different meshing approaches. 3D poisson problem run on two ranks. The labels indicate what multigrid method was used on what rank.

to reach a relative residual of 10^{-8} were recorded, and results from these tests are shown in Table 4.1. We can see that all three aggregation techniques perform similarly.

5. Conclusions. In this document we have discussed the implementation of a hybird multigrid method for partially structured meshes. Hybrid aggregation, building upon ideas from multigrid on HHG, aggregates regionally based on the local structure of the region. Interfaces between unstructured and structured meshes may require special treatment to ensure the resulting coarse mesh is conforming across the shared interface. The primary way to enforce interface agreement is for interface nodes in the same aggregate in the structured region to be aggregated together in the unstructured region.

A hybrid aggregation scheme has been implemented in MueLu. The implementation utilizes existing MueLu aggregation algorithms to construct aggregates based on the local regional structure. When used for piece-wise constant interpolation the hybrid aggregation method's performance is comparable to a purely algebraic or a purely geometric method. Hybrid aggregation grants the flexibility of algebraic multigrid along with the algorithmic advantages of structured multigrid with no negative impact on performance.

REFERENCES

[1] B. Bergen, G. Wellein, F. Hülsemann, and U. Rüde, *Hierarchical hybrid grids: Achieving teraflop performance on large scale finite element simulations*, Int. J. Parallel Emerg. Distrib. Syst., 22 (2007), pp. 311–329.
[2] B. K. Bergen and F. Hülsemann, *Hierarchical hybrid grids: Data structures and core algorithms for multigrid*, Numerical Linear Algebra With Applications, 11 (2004), pp. 279–291.
[3] W. Briggs, V. Henson, and S. McCormick, *A Multigrid Tutorial, Second Edition*, Society for Industrial and Applied Mathematics, second ed., 2000.
[4] J. Dendy, *Black box multigrid*, Journal of Computational Physics, 48 (1982), pp. 366 – 386.
[5] J. J. Hu, A. Prokopenko, C. M. Siefert, R. S. Tuminaro, and T. A. Wiesner, *MueLu multigrid framework*. http://trilinos.org/packages/muelu, 2014.
[6] J. Mandel, M. Brezina, and P. Vaněk, *Energy optimization of algebraic multigrid bases*, Computing, 62 (1999), pp. 205–228.
[7] P. Ohm and R. S. Tumin, *hybrid multigrid methods for nearly structured meshes*, Tech. Rep. SAND2017-1294R, Sandia National Laboratories, 2016. Center for Computing Research Summer Proceedings 2016.
[8] L. N. Olson, J. B. Schroder, and R. S. Tuminaro, *A general interpolation strategy for algebraic multigrid using energy minimization*, SIAM J. Scientific Computing, 33 (2011), pp. 966–991.
[9] A. Prokopenko, J. J. Hu, T. A. Wiesner, C. M. Siefert, and R. S. Tuminaro, *MueLu user's guide 1.0*, Tech. Rep. SAND2014-18874, Sandia National Labs, 2014.
[10] P. Vanek, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, COMPUTING, 56 (1996), pp. 179–196.
[11] P. Vaněk, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.

LAGRANGIAN PARTICLE METHODS FOR THE SHALLOW WATER EQUATIONS IN VARIED GEOMETRIES

NICHOLAS H. NELSEN* AND PETER A. BOSLER†

Abstract. We introduce a meshless Lagrangian particle-based numerical method inspired by vortex methods to solve the shallow water equations. Spatial discretization of differential operators on scattered particle distributions is performed using extensions of particle strength exchange and generalized moving least squares. Conservation of the discrete equations is emphasized; by construction, our method exactly conserves mass and maintains stationary flow states. We show that free space Green's functions for the Poisson equation can be used to recover the fluid flow velocity in the presence of challenging physical boundaries such as a moving wet/dry line, which hints at future applications in ocean modeling.

1. Introduction. Particle methods are a broad class of numerical schemes for the solution of partial differential equations (PDE). In these methods, the computational elements are particles—Dirac measures that carry system properties [12, 21]. To ease analysis and implementation in some applications, point particle approximations of the underlying field may be replaced by smoothed, regularized kernels that approximate the delta distribution in the limit of decreasing kernel radius [4].

Particles are advected along trajectories by the *flow map*,

$$\boldsymbol{X}(t;\,\cdot) : \mathbb{R}^d \to \mathbb{R}^d \tag{1.1a}$$

$$\boldsymbol{\xi} \mapsto \boldsymbol{X}(t;\boldsymbol{\xi})\,, \tag{1.1b}$$

where $\boldsymbol{X}(\tau;\boldsymbol{\xi}) = (X_1, X_2, \ldots, X_d)^T$ is the physical location of a particle at time $t = \tau$ that was initially located at the point $\boldsymbol{X}(0;\boldsymbol{\xi}) = \boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_d)^T$ at time $t = 0$. We note the plethora of notational choices for the flow map and refer the reader to common references, e.g., [9, 28, 45], for clarity. This reference frame is *Lagrangian* in the sense that it moves with the local flow velocity, as opposed to Eulerian fixed-grid schemes. The trajectories of particles $\mathbf{x}(t) = \boldsymbol{X}(t;\boldsymbol{\xi})$ are evolved in time along characteristics via the nonlinear ordinary differential equation (ODE)

$$\frac{d}{dt}\boldsymbol{X}(t) = \boldsymbol{u}(\boldsymbol{X}(t), t) \tag{1.2a}$$

$$\boldsymbol{X}(0) = \boldsymbol{\xi}\,, \tag{1.2b}$$

where the initial point $\boldsymbol{\xi}$ is called the Lagrangian particle marker, the position variable in Lagrangian space, and $\boldsymbol{u}$ is the velocity field. Once the Lagrangian formulation of the governing PDE has been written, a solution is often obtained from a method-of-lines discretization of the remaining ODE with any standard time integrator.

Purely Lagrangian particle methods allow for large time steps due to their lack of an advective CFL condition; advection error is independent of spatial resolution. Another advantage arising from the Lagrangian framework is the efficient placement of particles on compact subsets of the computational domain, for example, only on the locations where vorticity is nonzero [23] or, in this work, where mass is positive. However, computation of spatial derivatives can be more complicated in Lagrangian methods due to the generally nonuniform distribution of particles in physical space. Further, an unfortunate side effect of the Lagrangian frame is that particles tend to become distorted over time due to excessive flow strain. Frequently, such distortion leads to a loss of accuracy [6, 33]. Several successful

*Division of Engineering and Applied Science, nnelsen@caltech.edu
†Sandia National Laboratories, pabosle@sandia.gov

remedies have been proposed to correct this, including remeshing schemes [10], particle insertion and merging [2, 18], and radial basis function adaption [3].

Many modern numerical schemes incorporate a Lagrangian component, including smoothed particle hydrodynamics [32], immersed boundary methods [34], and vortex methods [21]. Vortex methods are concerned with the two and three-dimensional Euler vorticity equation of incompressible fluid flow. It is well known that in 2D, the flow kinematics are completely specified by an elliptic problem. Vortex particle methods use the Green's function of this elliptic problem along with the Biot–Savart law to invert the curl operator to recover velocity from vorticity. Thus, velocity is computed from the action of a nonlocal operator on a secondary variable instead of directly from the discretized momentum equation. Along with a rigorous convergence theory [19], vortex methods have had success modeling challenging flow phenomena, such as the vortex sheet simulations of Krasny [22, 23]. Comprehensive reviews of vortex and other particle methods may be found in [21, 26].

In this work, to better demonstrate their potential, we exploit the advantages of particles and vortex methods to develop a purely Lagrangian 1D vortex-type particle method for simulation of fluid flows in complex physical geometries. Governed by the shallow water equations, these problems are naturally expressed by the flow of mass-carrying particles, and this is reflected in the simple treatment of moving boundaries. Such an interpretation is also relevant to scientific fields that emphasize conservation, such as atmospheric and climate dynamics.

The remainder of this paper is as follows. The definition and properties of the shallow water equations are introduced in Sec. 2. In Sec. 3, we detail our meshfree Lagrangian particle method. Sec. 4 introduces the method of particle strength exchange for spatial discretization of differential operators, and we provide an alternative means of computing derivatives on scattered nodes via generalized moving least squares. Results are displayed and discussed in Sec. 5 for a variety of geometries and test problems in one spatial dimension. Finally, we conclude in Sec. 6 with a thorough summary and ideas for continued work.

2. Shallow Water Equations. The shallow water equations (SWE) are a system of hyperbolic conservation laws that model a 3D incompressible fluid under the *shallow water assumption*: vertical variability in the solution is asymptotically smaller than its horizontal variation. Hence, the full 3D flow is well approximated by a single thin layer of 2D fluid. The system is derived by applying the hydrostatic approximation to the 3D incompressible Euler equations of gas dynamics [45]. The resulting equations are two-dimensional in space and formally identical to the 2D isentropic Euler equations [27].

In advective form, which requires the solutions h and $\boldsymbol{u}$ to be sufficiently smooth, the SWE are defined by the system of nonlinear partial differential equations

$$\frac{Dh}{Dt} = \frac{\partial h}{\partial t} + \boldsymbol{u} \cdot \nabla h = -h(\nabla \cdot \boldsymbol{u}) \tag{2.1a}$$

$$\frac{D\boldsymbol{u}}{Dt} = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -f\boldsymbol{u}^{\perp} - g\nabla h\,, \tag{2.1b}$$

where $\boldsymbol{u}(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t))^T$ is the 2D velocity, $f(\mathbf{x})$ is the Coriolis parameter (which is nonzero for rotational flow problems), g is the gravity constant, and $h(\mathbf{x}, t)$ is the depth of the fluid. $D/Dt = \partial_t + \boldsymbol{u} \cdot \nabla$ denotes the material derivative and $\boldsymbol{u}^{\perp} = (-v, u)^T$ represents rotation of $\boldsymbol{u}$ by $\pi/2$. Note that the divergence of the 2D velocity, $\nabla \cdot \boldsymbol{u}$, is possibly nonzero. The model (2.1) may be interpreted as a thin fluid of constant density with variable depth over a 2D spatial domain. The free surface height of the fluid is denoted $s(\mathbf{x}, t)$. Bottom topography (or *bathymetry*) is defined by a function $s_B(\mathbf{x})$ so that $h(\mathbf{x}, t) = s(\mathbf{x}, t) - s_B(\mathbf{x})$, as shown in Fig. 2.1. The effect of a spatially varying bathymetry is the addition of a source

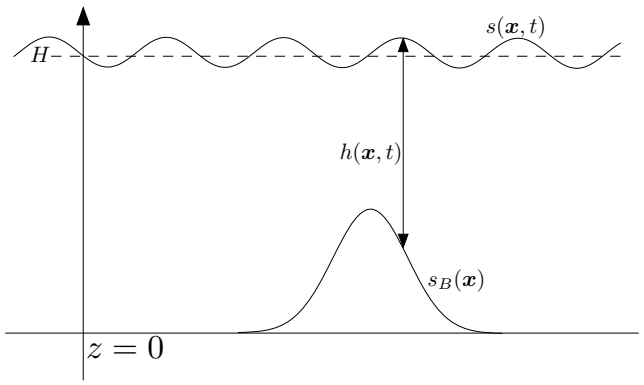


Fig. 2.1: Physical interpretation of the shallow water fluid domain. The vertical coordinate extends from $z = 0$ to $z = s$ at the fluid surface.

term $-g\nabla s_B$ in the momentum equation of the shallow water system (2.1). Note that for the common case of $s_B(\mathbf{x}) = 0$, we have $h(\mathbf{x}, t) = s(\mathbf{x}, t)$. The Lagrangian formulation of the SWE is obtained by supplementing Eqn. (2.1) with the flow map (1.2).

As with the Euler equations from which they are derived, the SWE have several conserved quantities. Let $\Omega \subseteq \mathbb{R}^2$ denote the problem domain; then, it is straightforward to derive the statements of mass and energy conservation,

$$\frac{d}{dt}\int_\Omega h\, dA = 0\,, \qquad \frac{d}{dt}\int_\Omega \frac{1}{2}\left(h|\boldsymbol{u}|^2 + gh^2\right) dA = 0\,, \tag{2.2}$$

respectively. The above conservation equations describe integral invariants. The SWE also have a material invariant quantity, the potential vorticity $\Xi = (\zeta + f)/h$, which satisfies $D\Xi/Dt = 0$, where $\zeta = (\nabla \times \boldsymbol{u}) \cdot \boldsymbol{e}_3$ is the relative vorticity in 2D [45]. Furthermore, the SWE admit gravity waves. Dispersion analysis of the linearized equations or applying the shallow water assumption to the full water wave equations [41] yields a wave speed $c = \sqrt{gh}$, which is frequently approximated by $c \approx \sqrt{gH}$ where the constant H is the mean fluid surface height, seen in Fig. 2.1.

The SWE have garnered much interest in atmospheric and oceanic modeling applications as well as the numerical analysis community, particularly in the development of improved numerical methods for property preservation and application-based modeling. A high-order finite volume scheme with the exact conservation property (C–property) maintained stationary flow states in [48]. In [49], model reduction was successfully applied to the SWE with realistic bathymetry profiles and passed a set of standard test problems. Within the related realm of atmospheric modeling, solution of the SWE on the sphere is of great interest [25, 45]. The ability of particle methods to inherently treat advection has lead to Lagrangian methods for tracer transport in atmospheric flows [7]. Further, Bosler *et al.* [6] implemented a vortex method for the barotropic vorticity equation on a rotating sphere as a promising prototype for the SWE. A suite of reference solutions in spherical geometries is available in [47], allowing for convenient verification and validation of these important physical problems. We aim to eventually apply our proposed numerical method to the rotating SWE on the sphere for atmosphere and climate simulations.

3. Description of Particle Method. The Lagrangian particle method (LPM) in this work aspires to combine the best traits of vortex methods and more general meshfree

schemes. We say *meshfree* in the sense that the nodes, or particles, are not connected in any predefined manner; instead, these unstructured nodes move with the local velocity field, and in problems involving shocks, particles may even be merged together or split apart. We now proceed to discuss the velocity integral in LPM.

3.1. Velocity Computation. As a 1D analog of vortex methods, LPM computes velocity using an integral transformation with kernel derived from the Green's function of the negative Laplacian, $-\Delta = -\partial_{xx}$. We use the Poisson equation in one dimension,

$$-\frac{d^2\phi}{dx^2} = f(x)\,, \tag{3.1}$$

with appropriate boundary conditions (BCs). Then, with the Green's function of the problem, the solution is

$$\phi(x) = \int_\Omega G(x,y) f(y)\, dy\ , \tag{3.2}$$

where G solves Eqn. (3.1) (in the weak sense) with $f(x) = \delta(x - \tilde{x})$, the delta distribution. G must be piecewise linear in the whole space or with Dirichlet BCs to satisfy Eqn. (3.1); hence, construction of these 1D Green's functions requires the solution of a four-by-four linear system determined by two BCs and two other relations, the continuity and jump conditions. Continuity at the source location $x = \tilde{x}$ implies

$$\lim_{x\to\tilde{x}^-} G(x,\tilde{x}) = \lim_{x\to\tilde{x}^+} G(x,\tilde{x})\,. \tag{3.3}$$

The jump condition due to the unit point source requires a unit discontinuity in the gradient of G, that is,

$$\lim_{x\to\tilde{x}^-} \frac{\partial}{\partial x} G(x,\tilde{x}) - \lim_{x\to\tilde{x}^+} \frac{\partial}{\partial x} G(x,\tilde{x}) = 1\,. \tag{3.4}$$

Inverting the resulting linear system yields the desired Green's function. We now apply this procedure to the periodic Poisson problem on $\Omega = [-\pi, \pi)$, which reads:

$$-\frac{\partial^2 G}{\partial x^2} = \delta(x-\tilde{x}) - \frac{1}{2\pi} \tag{3.5a}$$

$$G(-\pi,\tilde{x}) = G(\pi,\tilde{x}) \tag{3.5b}$$

$$\frac{\partial G}{\partial x} = \frac{\partial G}{\partial x}\,. \tag{3.5c}$$

Note that the differential equation is modified with a constant source term $-1/2\pi$ to satisfy the Neumann-type periodic boundary values in Eqn. (3.5). This implies that the condition $\int_\Omega -\partial_{xx} G\, dx = 0$ must hold [20] and that the Green's function will admit quadratic structure. We impose the additional equation $\int_\Omega G(x,\tilde{x})\, dx = 0$ to ensure a unique solution, yielding a six-by-six linear system. Solving, we obtain the piecewise smooth quadratic Green's function

$$G(x,\tilde{x}) = \frac{1}{4\pi}(x-\tilde{x})^2 - \frac{1}{2}|x-\tilde{x}| + \frac{\pi}{6}\,. \tag{3.6}$$

In the whole space, the corresponding function G is simpler. For all $x, \tilde{x} \in \mathbb{R}$, the free space Green's function is radially symmetric and is found to be

$$G(x,\tilde{x}) = G(|x-\tilde{x}|) = -\frac{1}{2}|x-\tilde{x}|\,. \tag{3.7}$$

Note that these Green's functions for $-\Delta$ in 1D lack the singularities exhibited by their counterparts in higher dimensions; hence, no special regularization is required. We remark that while our derivation of G is intrinsic to 1D, construction of Green's functions in higher dimensions proceeds differently but is well known (see, e.g., Evans [17]) for simple domains such as spheres or half-spaces.

Continuing, we decompose the velocity field u into its spatial mean ($\bar{u}$) and fluctuating ($\tilde{u}$) components,

$$u(x,t) = \bar{u}(t) + \tilde{u}(x,t)\,. \tag{3.8}$$

Now, suppose there exists a velocity potential ϕ defined by

$$\frac{\partial \phi}{\partial x} := \tilde{u}\,, \tag{3.9}$$

and denote the divergence of velocity $\nabla \cdot u$ in 1D as

$$\theta := \frac{\partial u}{\partial x} = \frac{\partial \tilde{u}}{\partial x}\,. \tag{3.10}$$

Combining Eqn. (3.9) and (3.10) yields a Poisson problem for the velocity potential,

$$\frac{\partial^2 \phi}{\partial x^2} = \theta\,, \tag{3.11}$$

which induces the Green's functions (3.6, 3.7). Therefore, once θ is known, $\tilde{u}$ may be computed from Eqn. (3.2) and (3.9).

3.2. Discrete Equations. We consider the 1D SWE in advective form:

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} = -h\frac{\partial u}{\partial x} \tag{3.12a}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -g\frac{\partial s}{\partial x} \tag{3.12b}$$

$$h(x,0) = h_0(x) \tag{3.12c}$$

$$u(x,0) = u_0(x)\,, \tag{3.12d}$$

where $t \in [0,\infty)$, $x \in \Omega \subseteq \mathbb{R}$, u is velocity, $s = h + s_B$ is the fluid surface height, s_B is the bathymetry profile, and $g > 0$ is the gravity constant. If posed on the whole space $\Omega = \mathbb{R}$, h and θ must have compact support, while on the periodic domain $\Omega = [-\pi, \pi)$, the conditions $u(-\pi,t) = u(\pi,t), h(-\pi,t) = h(\pi,t), \partial_x u(-\pi,t) = \partial_x u(\pi,t)$, and $\partial_x h(-\pi,t) = \partial_x h(\pi,t)$ must also be imposed. This continuous form of the SWE conserves mass, $\frac{d}{dt}\int_\Omega h\,dx = 0$, and in the periodic BC case, total divergence is zero: $\int_\Omega \theta\,dx = 0$.

To derive the Lagrangian form of Eqn. (3.12), we apply ∂_x to both sides of the momentum equation in (3.12) to yield a PDE for θ. However, the mean part $\bar{u}(t)$ is in the null space of ∂_x; it must be recovered by solving an additional ODE obtained by substituting the decomposition (3.8) into the momentum equation. Thus, after applying the flow map (1.1) and the change of variables $x(t) \to X(\xi;t)$, the Lagrangian form of the SWE is:

$$\frac{dx}{dt} = \bar{u} + \tilde{u} \tag{3.13a}$$

$$\frac{d\theta}{dt} = -\theta^2 - g\frac{\partial^2 s}{\partial x^2} \tag{3.13b}$$

$$\frac{d\bar{u}}{dt} = -\frac{d\tilde{u}}{dt} - g\frac{\partial s}{\partial x} \tag{3.13c}$$

$$\frac{dh}{dt} = -\theta h \tag{3.13d}$$

$$\frac{dJ}{dt} = \theta J\,, \tag{3.13e}$$

where the fluctuating part of velocity $\tilde{u}$ is

$$\tilde{u}(x,t) = \int_I -\frac{\partial G}{\partial x}\theta(y,t)\,dy\ , \tag{3.14}$$

G is the Green's function for $-\partial_{xx}$ on the collection of particles $I \subset \Omega$, and J is the Jacobian determinant of the flow map, $J(\xi,t) = \det(\partial_\xi X(t;\xi))$.

As a design choice, we consider a uniform initial particle distribution. Anisotropic initializations are also used in vortex methods [10,21], depending on the application. Given a computational domain $[a,b]$, we divide it into N cells and place a particle at the midpoint of each of these cells, $x_j|_{t=0} = a + (j-1/2)\Delta x\,,\ j = 1,\dots,N$, where the initial interparticle spacing is $\Delta x = (b-a)/N$. We define V_p as the associated particle volume element, initially $V_p(0) = \Delta x$. Thus, computation of integrals over the Lagrangian particle space in our method immediately amounts to midpoint rule quadrature. For particles $p = 1,2,\dots,N$, the discretized evolution ODE are

$$\frac{dx_p}{dt} = \bar{u} + \sum_{q=1}^{N} K_1(x_p - x_q)\theta_q V_q \tag{3.15a}$$

$$\frac{d\theta_p}{dt} = -\theta_p^2 - g\frac{\partial^2 s_p}{\partial x^2} \tag{3.15b}$$

$$\frac{d\bar{u}}{dt} = -\sum_{q=1}^{N} K_1(x_p - x_q)\frac{d}{dt}(\theta_q V_q) - g\frac{\partial s_p}{\partial x} \tag{3.15c}$$

$$\frac{dh_p}{dt} = -\theta_p h_p \tag{3.15d}$$

$$\frac{dV_p}{dt} = \theta_p V_p\,, \tag{3.15e}$$

where for t fixed, $\bar{u}(t)$ is constant for each p and the kernel

$$K_1(x) = -\frac{\partial G}{\partial x} := \begin{cases} \dfrac{1}{2}\operatorname{sgn}(x) - \dfrac{x}{2\pi}\,, & \Omega \text{ periodic} \\ \dfrac{1}{2}\operatorname{sgn}(x)\,, & \Omega = \mathbb{R}\,, \end{cases} \tag{3.16}$$

is a 1D analog of the Biot–Savart kernel. The sign function $\operatorname{sgn}(\cdot)$ is

$$\operatorname{sgn}(\mathrm{x}) = \begin{cases} \dfrac{x}{|x|}\,, & x \neq 0 \\ 0\,, & x = 0\,. \end{cases} \tag{3.17}$$

Further manipulating into *dilatation form*, the final form of the discrete Lagrangian shallow water equations in LPM read

$$\frac{dx_p}{dt} = \bar{u} + \sum_{q=1}^{N} K_1(x_p - x_q)Q_q \tag{3.18a}$$

$$\frac{dQ_p}{dt} = -gV_p \frac{\partial^2 s_p}{\partial x^2} \tag{3.18b}$$

$$\frac{d\bar{u}}{dt} = g \sum_{q=1}^{N} K_1(x_p - x_q) V_q \frac{\partial^2 s_q}{\partial x^2} - g \frac{\partial s_p}{\partial x} \tag{3.18c}$$

$$\frac{dM_p}{dt} = 0 \tag{3.18d}$$

$$\frac{dV_p}{dt} = Q_p \,, \tag{3.18e}$$

where $Q_p := \theta_p V_p$ is the particle dilatation and $M_p := h_p V_p$ is mass. Time discretization is a fourth order Runge–Kutta (RK4) scheme; it remains to discretize the right hand side spatial derivatives on the collection of particles. The dilatation form (3.18), inspired by [15], is preferred over the more standard divergence form (3.15) because it involves one less ODE and has superior conservation properties. This formulation exactly conserves mass at the discrete particle level: for all $t > 0$, $M_p(t) = M_p(0) = M_0$. Further, total divergence and particle volume are also conserved if $\sum_p Q_p(0) = 0$ and

$$\frac{d}{dt} \sum_p Q_p = -g \sum_p V_p \frac{\partial^2 s_p}{\partial x^2} = 0 \,. \tag{3.19}$$

Thus, vanishing global divergence in the discrete setting depends on the discretization of the Laplacian operator.

4. Spatial Derivatives on Particle Distributions. One challenge of meshfree approaches is the approximation of spatial derivatives. Since the particle distributions change in time and are most likely nonuniform and asymmetric, standard grid-based finite difference approximations, e.g., [39], may no longer be optimal. While finite differences have previously been used in vortex methods [2, 31], one of the ultimate goals of this work is high order accuracy; hence, we consider two alternative families of methods for computing derivatives on scattered data, particle strength exchange and moving least squares. To be consistent with the literature in this section, $h > 0$ denotes the average interparticle spacing and not the fluid depth variable $h(\mathbf{x}, t)$ in the SWE.

4.1. Particle Strength Exchange. The method of particle strength exchange (PSE) originated in the two-part set of papers [12, 13], in which integral operators for particle methods were developed to approximate the Laplacian appearing in advection-diffusion equations. PSE was significantly generalized by Eldridge, Leonard, and Colonius in [16], where arbitrary spatial derivatives

$$D^{\boldsymbol{\beta}} = \frac{\partial^{|\boldsymbol{\beta}|}}{\partial x_1^{\beta_1} \partial x_2^{\beta_2} \dots \partial x_d^{\beta_d}} \tag{4.1}$$

in the whole space $\mathbb{R}^d$ are approximated using appropriate kernel functions that satisfy a set of continuous moment conditions. Here, $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_d) \in \mathbb{N}_0^d$ is a multiindex, where $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. The PSE integral operator $P^{\boldsymbol{\beta}}$ approximates $D^{\boldsymbol{\beta}}$ as

$$P^{\boldsymbol{\beta}} f(\mathbf{x}) = \frac{1}{\varepsilon^{|\boldsymbol{\beta}|}} \int_{\mathbb{R}^d} (f(\mathbf{y}) \mp f(\mathbf{x})) \eta_\varepsilon(\mathbf{x} - \mathbf{y}; \boldsymbol{\beta}) \, d\mathbf{y} \,, \tag{4.2}$$

where $\eta(\cdot\,; \boldsymbol{\beta})$ is the kernel corresponding to the desired derivative and for $\varepsilon > 0$,

$$\eta_\varepsilon(\mathbf{x}) = \varepsilon^{-d} \eta(\mathbf{x}/\varepsilon) \,. \tag{4.3}$$

The kernel radius ε may be chosen as a scalar multiple of the desired support radius of the kernel or as a fractional power of the particle spacing, $h^{1/s}$, where $s > 1$. Further, the minus or plus sign in Eqn. (4.2) is chosen for symmetry reasons to ensure a conservative discretization. See [16] for details; in this work we are only concerned with the approximation of the Laplacian operator, for which the minus sign is appropriate. An r-th order accurate PSE approximation is constructed with Taylor expansions by ensuring the moment conditions

$$M^{\boldsymbol{\alpha}} = \begin{cases} (-1)^{|\boldsymbol{\beta}|}\boldsymbol{\beta}!\,, & \boldsymbol{\alpha} = \boldsymbol{\beta} \\ 0\,, & |\boldsymbol{\alpha}| = |\boldsymbol{\beta}| \ \wedge \ \boldsymbol{\alpha} \neq \boldsymbol{\beta} \\ 0\,, & |\boldsymbol{\alpha}| \in \{1, 2, \ldots, |\boldsymbol{\beta}| - 1\} \cup \{|\boldsymbol{\beta}| + 1, \ldots, |\boldsymbol{\beta}| + r - 1\} \end{cases} \tag{4.4a}$$

$$\int_{\mathbb{R}^d} |\mathbf{y}|^{|\boldsymbol{\beta}|+r} |\eta(\mathbf{y}; \boldsymbol{\beta})|\, d\mathbf{y} < \infty\,, \tag{4.4b}$$

where $M^{\boldsymbol{\alpha}} = \int_{\mathbb{R}^d} \mathbf{y}^{\boldsymbol{\alpha}} \eta(\mathbf{y}; \boldsymbol{\beta})\, d\mathbf{y}$. From [12], we have the error estimate

$$\left\| D^{\boldsymbol{\beta}} f - P^{\boldsymbol{\beta}} f \right\|_{L^2(\mathbb{R}^d)} \leq C\varepsilon^r \|f\|_{H^{r+2}(\mathbb{R}^d)}\,, \tag{4.5}$$

which is the optimal design rate of convergence for PSE. However, midpoint quadrature over the particles,

$$P_h^{\boldsymbol{\beta}} f_p = \frac{1}{\varepsilon^{|\boldsymbol{\beta}|}} \sum_q (f_q - f_p) \eta_\varepsilon(\mathbf{x}_p - \mathbf{x}_q; \boldsymbol{\beta}) V_q\,, \tag{4.6}$$

leads to the additional discretization error estimate [16],

$$\left\| P^{\boldsymbol{\beta}} f - P_h^{\boldsymbol{\beta}} f \right\|_{L^2(\mathbb{R}^d)} \leq \tilde{C} \frac{h^s}{\varepsilon^{s+|\boldsymbol{\beta}|-1}} \|f\|_{H^s(\mathbb{R}^d)}\,, \tag{4.7}$$

which requires $\varepsilon/h > 1$, that is, particles must overlap for the error to decay. Further, $f \in H^s(\mathbb{R}^d)$ must either be periodic or have compact support and $\eta \in W^{s,1}(\mathbb{R}^d)$ to observe optimal rates (see [1, 17] for a review of Sobolev spaces). This restrictive set of functions is a limitation of PSE; alternative methods such as DC–PSE (Sec. 4.1.1) or moving least squares (Sec. 4.2) can perform optimally for a larger class of functions. Yet by construction, PSE discretizations have the advantage of being *numerically conservative* in the sense that the sum over all particles of a PSE operator acting on some function f is identically zero.

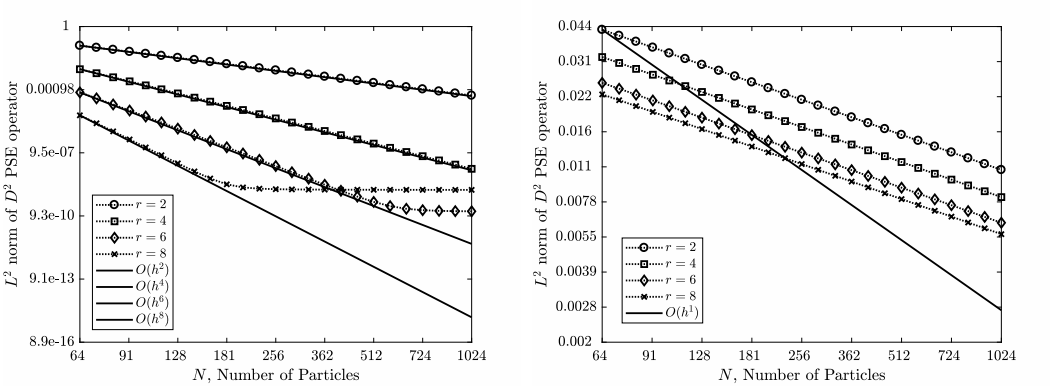
PSE kernels are commonly constructed from Gaussians, which are in $W^{\infty,1}(\mathbb{R}^d)$. In 1D, we use the kernel template

$$\eta(x) = \frac{x^{(\beta \bmod 2)}}{\sqrt{\pi}} \left(\sum_{j=0}^{r/2-1} a_j x^{2j} \right) e^{-x^2} \tag{4.8}$$

for $r = 2, 4, 6$, and 8-th order kernels (see [16] for explicit formulas), where the constants a_j are to be determined from a system of $r/2$ linear equations in Eqn. (4.4). Hence, the Laplacian (equivalent to D^2 in 1D) may be approximated by

$$\Delta_\varepsilon f(x) = \varepsilon^{-2} \int_{\mathbb{R}} (f(y) - f(x)) \eta_\varepsilon(x - y; 2)\, dy\,, \tag{4.9}$$

where $\Delta_\varepsilon \to \Delta$ as $\varepsilon \to 0$. This theoretically should provide spectrally accurate quadrature for appropriate f when discretized [16]. However, we now show that PSE has difficulty



(a) $f(x) = \exp(-\pi x^2)$ (b) $f(x) = \cos(x)$

Fig. 4.1: Convergence study of standard PSE approximation to $D^2 f$ with $\varepsilon/h = 1.8$.

achieving optimal convergence rates in 1D test cases for D^2; this behavior was also identified in [37].

In Fig. 4.1, the PSE integral approximation to the second derivative D^2 is applied to two functions on the interval $[-\pi, \pi]$ with uniform particle spacing at constant ratio $\varepsilon/h = 1.8$. The results that follow are also observed on randomly initialized particle distributions. In the first case Fig. 4.1a, $f(x) = \exp(-\pi x^2)$ satisfies the conditions for optimal convergence and this is observed for kernels of order $r = 2, 4$. However, the dependence on the kernel-to-particle spacing ratio ε/h (here $\varepsilon/h = 1.8$) is apparent at high kernel orders, where the error saturates due to the discretization estimate (4.7). Further, we do not observe convergence of the function $f(x) = \cos(x)$ in Fig. 4.1b; this is expected since $\cos(x) \notin H^s(\mathbb{R})$ for any $s \geq 0$.

4.1.1. Discretization Correction. Schrader in [37] extended the capabilities of the general derivative treatment of Eldridge *et al.* [16] by introducing discretization corrected particle strength exchange (DC–PSE) operators. The distinction here is that the operators of DC–PSE exactly satisfy discrete moment conditions instead of the continuous ones in Eqn. (4.4). As a result, the discretization error in Eqn. (4.7) is substantially reduced, albeit at the expense of exact conservation and additional computational cost. Indeed, on nonuniform or asymmetric particle distributions, a linear system must be solved at each particle location to derive the spatially dependent discretization corrected kernel. Despite these shortcomings, DC–PSE is advantageous over standard PSE due to its improved accuracy near boundaries and less stringent parameter choice limitations.

The discrete moment conditions nearly parallel those in Eqn. (4.4). We have

$$M_h^{\boldsymbol{\alpha}}(\mathbf{x}) = \frac{1}{\varepsilon^d} \sum_{\{q:\, \mathbf{x}_q \in B(\mathbf{x}, R_c)\}} \left(\frac{\mathbf{x} - \mathbf{x}_q}{\varepsilon}\right)^{\boldsymbol{\alpha}} \eta^{DC}\left(\frac{\mathbf{x} - \mathbf{x}_q}{\varepsilon}; \boldsymbol{\beta}\right) V_q \,, \tag{4.10}$$

subject to

$$M_h^{\boldsymbol{\alpha}} = \begin{cases} (-1)^{|\boldsymbol{\beta}|}\boldsymbol{\beta}! \,, & \boldsymbol{\alpha} = \boldsymbol{\beta} \\ 0 \,, & \boldsymbol{\alpha} \neq \boldsymbol{\beta} \ \wedge \ \alpha_{min} \leq |\boldsymbol{\alpha}| \leq |\boldsymbol{\beta}| + r - 1 \,, \end{cases} \tag{4.11a}$$

$$M_h^{\boldsymbol{\alpha}} < \infty \,, \quad |\boldsymbol{\alpha}| = |\boldsymbol{\beta}| + r \,, \tag{4.11b}$$

where $B(\mathbf{x}, R_c)$ is the ball of radius R_c centered at $\mathbf{x} \in \mathbb{R}^d$, r is the desired order of accuracy, and α_{min} equals zero if $|\boldsymbol{\beta}|$ is odd and one otherwise. The DC–PSE kernel template involves a more general polynomial correction factor than does PSE,

$$\eta^{DC}(\mathbf{y}; \mathbf{x}, \boldsymbol{\beta}) = \left(\sum_{|\boldsymbol{j}|=\alpha_{min}}^{|\boldsymbol{\beta}|+r-1} a_j(\mathbf{x}) \mathbf{y}^{\boldsymbol{j}} \right) e^{-|\mathbf{y}|^2}, \tag{4.12}$$

where the coefficients $a_j(\mathbf{x})$ are to be determined. In 1D, substitution of Eqn. (4.12) into Eqn. (4.10) along with the conditions (4.11) leads to the linear system of $\beta + r - \alpha_{min}$ equations

$$\sum_{j=\alpha_{min}}^{\beta+r-1} a_j(x_p) W_{\alpha,j}(x_p) = M_h^{\alpha}(x_p) \tag{4.13}$$

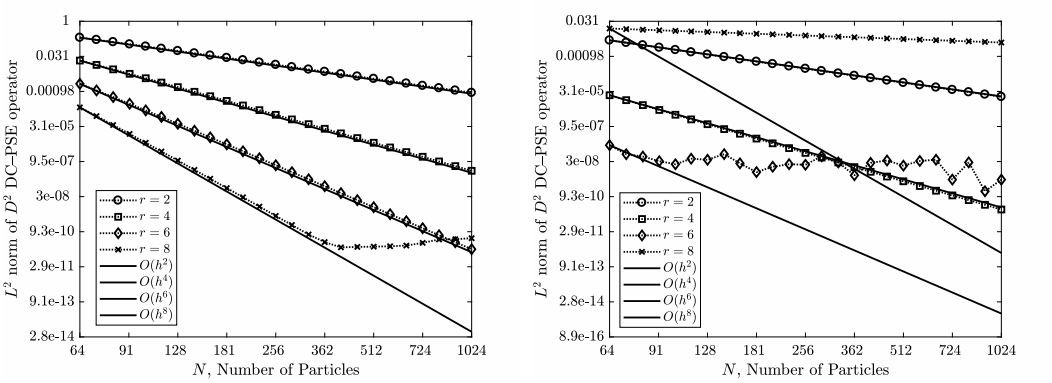
for each particle p, where the weights are

$$W_{\alpha,j}(x) = \frac{1}{\varepsilon^{1+\alpha+j}} \sum_{\{q:\, |x-x_q|<R_c\}} (x - x_q)^{j+\alpha} e^{-\left(\frac{x-x_q}{\varepsilon}\right)^2} V_q \,. \tag{4.14}$$

The DC–PSE approximation is then

$$P_h^{DC,\boldsymbol{\beta}} f_p = \frac{1}{\varepsilon^{|\boldsymbol{\beta}|}} \sum_{\{q:\, \mathbf{x}_q \in B(\mathbf{x}, R_c)\}} (f_q - f_p) \eta_{\varepsilon}^{DC}(\mathbf{x}_p - \mathbf{x}_q; \boldsymbol{\beta}) V_q \,, \tag{4.15}$$

Our implementation of DC–PSE yields optimal convergence rates, irrespective of the computational domain boundaries; this holds even without using a cutoff radius R_c, i.e., $R_c = \infty$. DC–PSE proves to be a viable alternative to the one-sided PSE operators described in [16], especially for approximating the Laplacian; we find that one-sided kernels for D^2 did not converge for a variety of cases. DC–PSE is also sensitive to the ratio ε/h, but less



(a) $f(x) = \exp(-\pi x^2)$ (b) $f(x) = \cos(x)$

Fig. 4.2: Convergence study of DC–PSE approximation to $D^2 f$ with $\varepsilon/h = 1.8$.

so than standard PSE. DC–PSE even allows for non-overlapping ratios of $\varepsilon/h \leq 1$; in this scenario, the method approaches finite difference stencils [37]. For high-order kernels, we

observe that increasing ε/h improves accuracy and consistently leads to optimal convergence rates. In contrast, increasing the ratio in standard PSE does not in general improve accuracy. Compared to Fig. 4.1a, Fig. 4.2a shows that DC–PSE outperforms standard PSE in terms of convergence under identical conditions. Further, we observe in Fig. 4.1b that the L^2 error corresponding to the DC–PSE second derivative approximation for $f(x) = \cos(x)$ decays as $O(\varepsilon^r)$ for $r = 2, 4$, demonstrating that a larger collection of functions is admissible for DC–PSE. Overall, however, DC–PSE still struggles with high-order kernels because of inherent linear system ill-conditioning.

4.2. Moving Least Squares. The method of moving least squares (MLS) is a meshless approach for function approximation on scattered data (see the text of Wendland [46] for a rigorous treatment). Originally formulated in the seminal work of Lancaster [24] as a means of surface reconstruction in high dimensions, recently much interest has been given to the use of MLS in the numerical solution of partial differential equations. In [35], a MLS discretization of derivatives was implemented for the challenging compound KdV–Burgers' equation. A compatible meshfree scheme using a high-order staggered discretization in [43] was used to solve steady Stokes flows, while in [42] a compact-stencil version of MLS was introduced that generalizes the well-known compact finite difference stencils to arbitrary point sets. We describe the formulation of MLS in 1D; generalization to higher dimensions is straightforward.

MLS is an extension of classical weighted least squares minimization and provides a direct optimal estimation of an unknown function from known field values. Given a set of points $\{x_j\}_{j=1}^N \subset \Omega \subseteq \mathbb{R}$ with associated data $\{\varphi(x_j)\}_{j=1}^N$, the goal is to find the spatially-dependent coefficients $\mathbf{a}^{opt}(x) = \{a_j(x)\}_{j=1}^R$ that satisfy the approximation

$$\varphi(x) \approx \varphi_h(x) = \sum_{j=0}^{R} a_j(x) p_j(x) = \mathbf{p}^T(x)\mathbf{a}^{opt}(x) \tag{4.16}$$

in the sense of a local polynomial reproduction [46], where $\varphi \in C^{m+1}(\Omega)$ is the unknown function and $\mathbf{p} = \{p_j\}_{j=1}^R$ is a basis for the finite-dimensional space $\Pi_m(\mathbb{R})$ of polynomials of degree less than or equal to m. In fact, the coefficient vector $\mathbf{a}^{opt}(x)$ solves the weighted ℓ^2-optimization problem

$$\mathbf{a}^{opt} = \underset{\mathbf{a}\in\mathbb{R}^{\dim \Pi_m(\mathbb{R})}}{\arg\min} \sum_{j=1}^{N} \left|\varphi(x_j) - \mathbf{p}^T(x_j)\mathbf{a}(x_j)\right|^2 W(|x - x_j|)\,, \tag{4.17}$$

where $x \in \Omega$ and $W = W(|x|)$ is the positive weight. The well known solution is

$$\mathbf{a}^{opt}(x) = \mathbf{M}^{-1}(x) \sum_{j=1}^{N} \mathbf{p}^T(x_j) W(|x - x_j|)\varphi(x_j)\ , \tag{4.18}$$

where $\mathbf{M}(x) = \sum_j \mathbf{p}^T(x_j)W(|x - x_j|)\mathbf{p}(x_j)$. As in DC–PSE, a small weighted least squares problem is solved for each target point, usually by matrix factorization or pseudoinverse.

4.2.1. Generalized Moving Least Squares. In MLS, derivatives of φ are approximated by taking derivatives of the shape functions generated by the MLS process. By contrast, the action of target functionals is directly recovered from the data in the generalized moving least squares (GMLS) approach. We refer the interested reader to Mirzaei [30] for detailed analysis of GMLS and its relation to the so-called "diffuse derivative."

Given a MLS approximation $\varphi_h = \mathbf{p}^T(x)\mathbf{a}(x)$ (4.16), the GMLS derivative approximation for φ_h is defined as

$$\left(D_h^\beta \varphi\right)(x) := \left(D^\beta \mathbf{p}(x)\right)^T \mathbf{a}(x)\,, \tag{4.19}$$

and from [30, 42] we have the optimal convergence rate

$$\left\| D^\beta \varphi - D_h^\beta \varphi \right\|_{L^\infty(\Omega)} \leq \tilde{C}(m) h_{fill}^{m+1-\beta} \|\varphi\|_{W^{m+1,p}(\Omega)}\,, \tag{4.20}$$

where the fill distance h_{fill} describes the radius of the largest ball not containing any data sites [46]. In practice, $1/N$ or the average particle spacing h may be used as the characteristic length instead of the exact value for h_{fill}.

4.2.2. Generalized Moving Least Squares with Enrichment. We now further generalize the GMLS approach by designing a function space admitting piecewise continuous polynomials. Motivated by the need to treat moving boundaries in Sec. 5.3, we seek function reconstruction over the space of polynomials defined piecewise on sets Ω_i such that $\Omega_i \cap \Omega_j = \varnothing$ for all $i \neq j$. Our approach is to construct approximations from the space of functions

$$\mathcal{P}_m(\mathbb{R}) := \{\chi_I p + \chi_B q : p, q \in \Pi_m(\mathbb{R})\}\,, \tag{4.21}$$

where we define the usual indicator

$$\chi_D(x) = \begin{cases} 1\,, & x \in D \\ 0\,, & x \notin D \end{cases}\,. \tag{4.22}$$

In this work, the sets I and B are disjoint so that $\chi_B(x) = 1 - \chi_I(x)$. For the problems discussed later in Sec. 5.3, I will denote the interior of a domain Ω and B the points on the boundary $\partial\Omega$. For notational brevity we denote this approach as EGMLS and note that a more general framework may be found in [44]. This formulation is just generalized moving least squares with the basis enriched with discontinuous polynomials. EGMLS handles the difficulty of computing derivatives in the neighborhood of a discontinuity or non-differentiable point. Setting $I = \Omega$ reduces the method to GMLS with the standard polynomial basis.

The specifics of our EGMLS implementation are now detailed. We use the family of radially symmetric weight kernels

$$W(|x-y|) = \begin{cases} \left(1 - \dfrac{|x-y|}{\epsilon}\right)^p\,, & |x-y| < \epsilon \\ 0\,, & |x-y| \geq \epsilon \end{cases} \tag{4.23}$$

where $p > 0$, y is the center, and ϵ is the support radius. In this work, $p = 6$. The stencil of data location points x_j is chosen to be of size $2s+1$, where

$$s = \text{ceil}(m/2) + 1\,. \tag{4.24}$$

This choice of s guarantees a sufficient number of points in the stencil for unisolvency over the set in Eqn. (4.21). To ensure polynomial unisolvency conditions [46] are met at *each stage* of the computation, we introduce an adaptive kernel radius $\epsilon = \epsilon(s)$. Trask *et al.* [43] similarly adjusted the MLS weight function for improved multiresolution simulation. Specifically, the support radius ϵ is adaptively refined for each particle x_j based on the average spacing

between neighboring particles in the stencil of interest. That is, given the average particle spacing $h = \frac{1}{2s}\sum_j (x_{j+1} - x_j)$, the variable support ϵ is defined to be

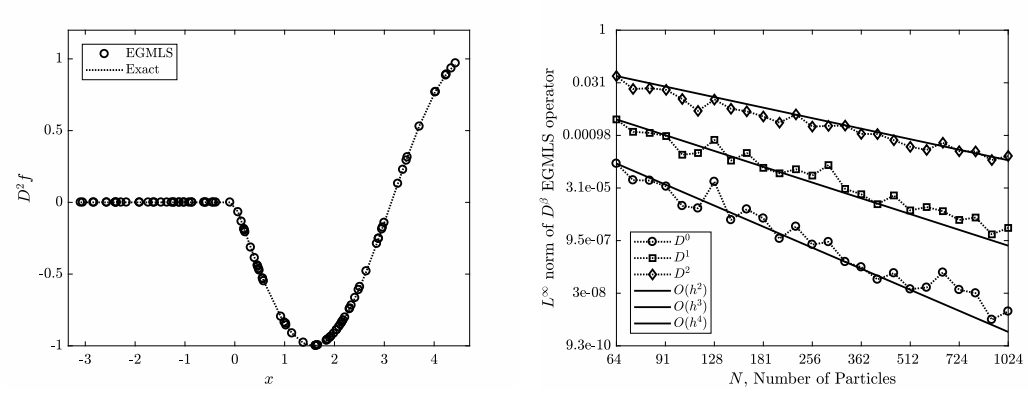
$$\epsilon(s) = \left(2s + \frac{1}{2}\right) h \,. \tag{4.25}$$

To improve the conditioning of the matrix inversion in Eqn. (4.18), our polynomial bases $\mathbf{p}, \mathbf{q}$ for $\Pi_m(\mathbb{R})$ in Eqn. (4.21) consist of Taylor monomials shifted and scaled:

$$p_{k;i} = \frac{1}{k!}\left(\frac{x - x_i}{\epsilon}\right)^k \,, \tag{4.26}$$

and similarly for $q_{k;i}$ [30, 42]. Conveniently, for each $x_i \in \Omega$, $D^\beta \mathbf{p}(x_i) = \epsilon^{-\beta}$. Thus,

$$D_h^\beta \varphi(x_i) = \epsilon^{-\beta} \mathbf{a}(x_i) \,. \tag{4.27}$$



(a) EGMLS approximation of $D^2 f$ with $N = 100$ (b) L^∞ error for various derivative orders β

Fig. 4.3: EGMLS applied to $f(x)$ in Eqn. (4.28) with $m = 3$ on uniformly distributed random particles. The "kink" interface at $x = 0$ in $D^2 f$ is correctly captured (Fig. 4.3a). In Fig. 4.3b, the particles are resampled at each level of refinement.

As a simple test, we consider the piecewise smooth function $f \in C^0(\Omega)$:

$$f(x) = \begin{cases} \sin(x) \,, & x > 0 \\ 0 \,, & x \leq 0 \end{cases} \tag{4.28}$$

on points randomly sampled from the asymmetric domain $\Omega = [-\pi, 3\pi/2]$. Here, $I = \{x : x > 0\}$. Fig. 4.3a suggests that EGMLS accurately captures the non-differentiable interface at $x = 0$ of $D^2 f$ in the weak sense. We observe the same optimal rate $O(h^{m+1-\beta})$ of standard GMLS (4.20) in Fig. 4.3b. For functions with less than the required C^{m+1} regularity of MLS and GMLS (e.g., solutions to PDE with large gradients), EGMLS may be an appropriate alternative for particle derivative approximations.

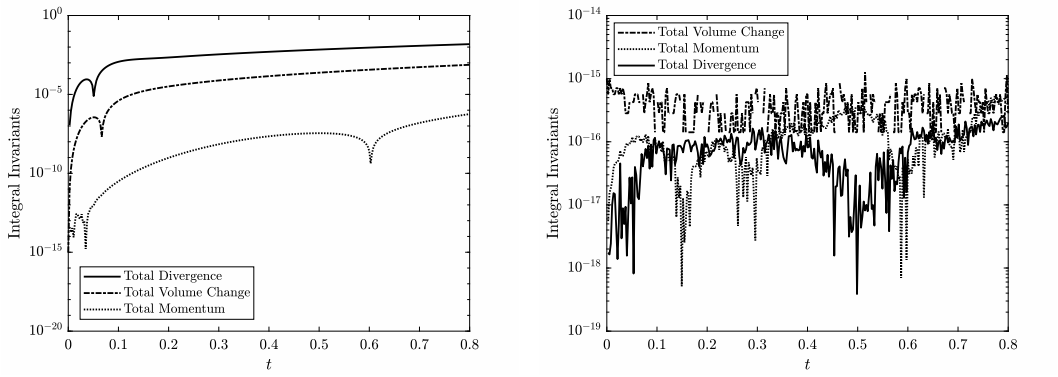
5. Numerical Results. We pose the 1D shallow water equations (3.12) on two distinct domains, the whole real line $\mathbb{R}$ and the 2π-periodic domain $[-\pi, \pi)$. Within these spaces, we imitate physical geometries by changing the bathymetry profiles. These source terms s_B can have a profound effect on the behavior of solutions; for example, by prescribing a parabolic topography to simulate an ocean basin, complex moving boundaries with wetting and drying phenomena arise (Sec. 5.3). We apply EGMLS, PSE, and DC–PSE in LPM for derivative computation of the fluid surface Laplacian. Note that for PSE or DC–PSE, the ODE for particle dilatation in (3.18) becomes

$$\frac{dQ_p}{dt} = -\frac{g}{\varepsilon^2}\sum_{q=1}^{N}\big[V_pM_q - V_qM_p + (V_pV_q)(s_B(x_q) - s_B(x_p))\big]\eta_\varepsilon(x_p - x_q; 2)\,. \tag{5.1}$$

In all LPM simulations, gravity is set to $g = 9.80616$ and time integration is with RK4.

5.1. Whole Real Line. We approximate the SWE on the unbounded domain $\mathbb{R}$ by using an interval $[a, b]$ large enough such that shallow water waves do not reach these numerical boundaries in the alloted simulation time t_{max}. The free space Green's function and one-sided derivative stencils (for the EGMLS operator D_h^2) are utilized to enforce the unbounded interpretation of this domain. Our PSE-type discretizations were derived only for the whole space and hence no unique treatment is required.

We test the ability of LPM to conserve integral invariants of the SWE: total momentum $\sum_p M_p(t)u_p(t)$, total divergence $\sum_p Q_p(t)$, and total relative volume change $([\sum_p V_p(t)] - b + a)/(b - a)$. The simulations are initialized with $t_{max} = 0.8$, $\Delta t = 0.0027$, $N = 150$ particles, $m = 4$ for EGMLS, $r = 4$ with $\varepsilon/\Delta x = 2$ for PSE/DC–PSE, $s_B = 0$, $u_0 = 0$, and $h_0(x) = 0.75 + 0.075\exp\big(-100\pi x^2/9\big)$ on the computational domain $\Omega := [a, b] = [-\pi, \pi]$. The small amplitude initial data $h_0(x)$ emphasizes the shallow water assumption and avoids shocks.



(a) Integral invariants from LPM–EGMLS

(b) Integral invariants from LPM–PSE

Fig. 5.1: Time evolution of SWE invariants. Values grow in time with LPM–EGMLS (Fig. 5.1a) while are conserved to double precision with LPM–PSE (Fig. 5.1b).

Table 5.1 and Fig. 5.1b reaffirm the exact conservation of standard PSE; total particle volume, momentum, and divergence all maintain a steady machine precision error over the entire simulation duration. On the other hand, the invariant quantities using LPM–EGMLS display a drift over time (Fig. 5.1a). From Table 5.1, we observe that LPM–DCPSE conserves

Quantity	EGMLS	PSE	DC–PSE
Volume Change	7.52E-04	2.83E-16	1.88E-04
Momentum	5.68E-07	5.33E-16	9.61E-16
Divergence	1.53E-02	1.99E-16	3.06E-03

Table 5.1: Values of integral invariants volume, momentum, and divergence at final time $t_{max} = 0.8$. PSE conserves all three quantities to machine precision, whereas only momentum is conserved with DC–PSE and none for EGMLS.

momentum to double precision over the course of the simulation, but particle volume and divergence only to a similar threshold as in LPM–EGMLS.

While our unbounded domain approximation is a useful testbed geometry for comparing properties of each spatial derivative discretization technique, more complex flow pattens such as wave-wave interactions cannot be observed; we now turn to a periodic domain.

5.2. Periodic Domain. The periodic domain $[-\pi, \pi)$ is modeled by utilizing the periodic velocity kernel K_1 in Eqn. (3.16) along with special modifications for EGMLS. Specifically, at each time step LPM checks if any particle has left the domain; if so, that particle is shifted back into the domain appropriately. For each particle p, our EGMLS routine maintains particle-centered stencils by using the periodic image of the neighbors of p, when necessary. While the PSE-type discretizations have not yet been extended to periodic boundary conditions, we still test such methods by using the periodic version of kernel K_1 and setting t_{max} small enough such that waves are not propagated to the edges of the computational domain.

A scheme is said to have the exact C–property if it is exact (i.e., to machine precision) for stationary solutions. This is challenging for hyperbolic systems with source terms, such as SWE with the term s_B. We conduct an exact C–property test on the periodic interval with the following initializations: $s(x,t) = s_0(x) = 0.75$, $u(x,t) = u_0(x) = 0$, $s_B(x) = 0.5\exp\left(-\pi x^2\right)$, and the remaining parameter choices maintained the same as those outlined in Sec. 5.1.

L^2 Error Quantity	EGMLS	PSE	DC–PSE
Surface: $\|s(\cdot, t_{max}) - s_0\|_{L^2(\Omega)}$	3.46E-15	3.68E-17	1.11E-11
Velocity: $\|u(\cdot, t_{max}) - u_0\|_{L^2(\Omega)}$	4.04E-15	1.24E-16	8.17E-11

Table 5.2: L^2 error of stationary surface height and velocity at final time $t_{max} = 0.8$. Maintaining the steady solution to near machine precision, LPM with EGMLS and PSE demonstrate the exact C–property.

Table 5.2 suggests that both LPM–PSE and LPM–EGMLS preserve the stationary "lake at rest" solution [5], while LPM–DCPSE introduces some numerical error.

Additional LPM–EGMLS studies were conducted on the periodic domain that compared solutions to those of a reference high-resolution central-in-space finite difference scheme; LPM–EGMLS was in good agreement with the reference for all of the small-amplitude initial conditions tested, even during wave-wave interactions.

5.3. Ocean Basin. Of great significance in physical oceanography is the free surface flow of water in lakes or ocean basins. Simple models of ocean circulation have lead to

great advances in our understanding of observable phenomena. For example, the Stommel model [38] is a Helmholtz equation with sinusoidal forcing that provides an explanation for the intensification of Western boundary layer development in wind-driven circulations such as the Gulf Stream [29]. The SWE have also been used to model the propagation of tsunamis in the ocean [41]. In this section, we investigate the flow of water in a parabolic basin and assess the ability of the free-space version of kernel K_1 to handle nontrivial physical geometries.

Following Thacker [40] and other sources of analytical SWE solutions [11, 36, 47], we prescribe the parabolic bathymetry profile

$$s_B(x) = -A_0 \left(1 - \frac{(x - x_c/2)^2}{L^2}\right), \tag{5.2}$$

where A_0 is the peak parabola amplitude, x_c is the offset, and L is the horizontal extent from $x = 0$ to the shoreline $s_B = 0$ (Fig. 5.2). From [14], the exact solution for an oscillating

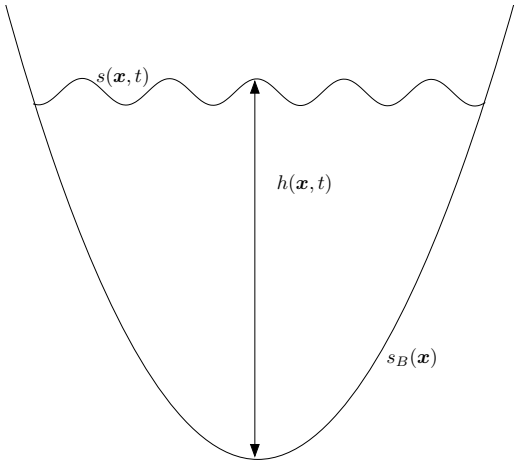


Fig. 5.2: Parabolic ocean basin geometry for the SWE defined by source term s_B.

1D linear surface flow governed by SWE with s_B of the form (5.2) is

$$h(x,t) = \begin{cases} -A_0 \left[\left(\frac{1}{L}\left(x - \frac{x_c}{2}\right) + \frac{1}{2L}\cos\left(\frac{\sqrt{2gA_0}}{L}t\right)\right)^2 - 1 \right], & x_1(t) \le x \le x_2(t) \\ 0, & \text{else} \end{cases} \tag{5.3}$$

$$u(x,t) = \begin{cases} \frac{\sqrt{2gA_0}}{2L}\sin\left(\frac{\sqrt{2gA_0}}{L}t\right), & x_1(t) \le x \le x_2(t) \\ 0, & \text{else} \end{cases}, \tag{5.4}$$

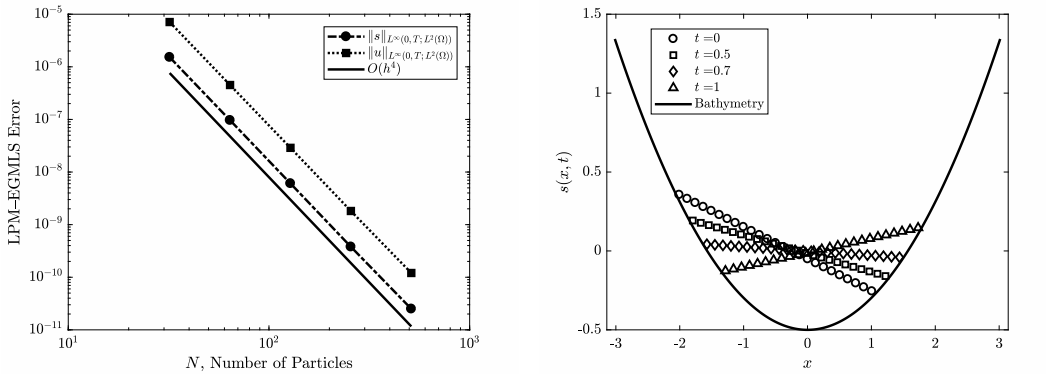
where the time-varying wet-dry interface locations are

$$x_1(t) = -\frac{1}{2}\cos\left(\frac{\sqrt{2gA_0}}{L}t\right) - L + \frac{x_c}{2}, \tag{5.5a}$$

$$x_2(t) = -\frac{1}{2}\cos\left(\frac{\sqrt{2gA_0}}{L}t\right) + L + \frac{x_c}{2}. \tag{5.5b}$$

Initial conditions for this initial value problem are $h(x,0) = h(x,t)|_{t=0}$ and $u(x,0) = 0$. Accurately capturing the continuously evolving non-differentiable transition from wet fluid ($h > 0$) to dry shore ($h = 0$) in this hyperbolic system is a salient challenge for computational schemes.

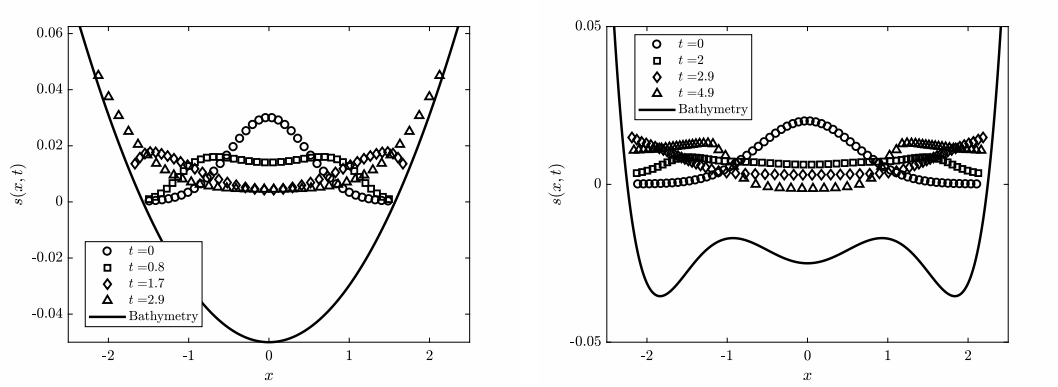
We perform a convergence study on this problem with LPM. The problem is only discretized on the domain $I = \{x : h(x,\cdot) > 0\}$, the particles with mass. Test parameters were $A_0 = 0.5$, $L = \pi/2$, $x_c = 0$, $T := t_{max} = 6$, and $N \in \{32, 64, 128, 256, 512\}$. As the spatial resolution increased, we maintained the ratio $\Delta t/\Delta x = 0.5$ to determine the refined time step.



(a) $L^\infty(0,T;L^2(\Omega))$ error of $s(x,t)$ and $u(x,t)$ (b) Snapshots of oscillating linear surface solution

Fig. 5.3: Oscillating linear surface problem results for LPM–EGMLS: fourth order convergence (Fig. 5.3a) of $s(x,t)$ and $u(x,t)$ with $m = 2$ and solution snapshots (Fig. 5.3b) of the oscillating surface $s(x,t)$ with $m = 3$.

For EGMLS polynomial order $m = 2$, we observe in Fig. 5.3a fourth order spatial convergence (taking the infinity norm in time) to the analytical solution Eqns. (5.3, 5.4) for $s(x,t)$ and $u(x,t)$. Although the midpoint quadrature for the velocity integral is formally second order, this specific choice of initial value problem admits zero divergence θ since for fixed time t, $u(x,t)$ is piecewise constant (5.4). Thus, $\tilde{u} = 0$ and the only velocity contribution is from the mean part $\bar{u}(t)$. Since the EGMLS formulation is exact for linear functions by the polynomial reproduction property, the observed fourth order rate is actually the order of the RK4 time integration scheme. Snapshots of the LPM solution are shown in Fig. 5.3b; the surface sloshes along the basin walls and remains linear in time, as expected. This tide-like inundation behavior is also observed for more complex initial surfaces, including a Gaussian drop in the same parabolic basin and in a bumpy sea bed defined by a sixth order polynomial profile (Fig. 5.4). LPM naturally handles challenging moving boundaries that arise.



(a) Parabolic basin (b) Sextic basin

Fig. 5.4: Gaussian drop initial surface results for LPM–EGMLS: solution snapshots of $s(x,t)$ in parabolic basin (Fig. 5.4a) and sextic polynomial basin (Fig. 5.4b) with $m = 2$. In both cases, the waves exhibit wetting and drying. We note that the visual gap at the wet/dry interface between the bathymetry and nearest particles in Fig. 5.4a and Fig. 5.4b is only an artifact of our cell-centered particle initialization in Sec. 3.2; no particle is actually placed on the interface itself.

6. Conclusion. This work has developed a mass-conservative 1D analog of a vortex particle method for the shallow water equations posed on varied domains. Adjustment to the bathymetry source term s_B allowed for models of physically relevant geometries and complex moving boundaries. The method's simple treatment of wet/dry transitions, based on only tracking the particles with positive mass, was successfully demonstrated in a parabolic basin with an oscillating linear surface flow and in parabolic and sextic basins with a Gaussian drop. Indeed, the results in Section 5 suggest that the nonlocal Green's function approach to velocity computation in LPM is appropriate for simulating ocean-related flows.

Moreover, we have presented two techniques from the literature, particle strength exchange and moving least squares, and their extensions for approximating derivatives in particle methods. Standard PSE is optimal for applications in which conservation is a high priority, such as climate simulations. While the treatment of numerical boundaries is challenging for PSE, coupling with DC–PSE near these locations could serve as a remedy. In problems requiring more general spaces of functions or difficult interface definitions, our results suggest that the new EGMLS approach is the superior choice. However, we acknowledge that the extension of our specific discontinuous polynomial enrichment approach to dimensions $d > 1$ is a significant challenge. For example, construction of the required indicator functions is not as obvious in higher dimensions and would likely involve sophisticated algorithms from computational geometry.

To conclude the discussion of the 1D case, we remark that LPM was applied to standard linear advection-diffusion and viscous Burgers' equation problems and showed close agreement with reference solutions. Our future work on the 2D SWE will incorporate treecodes for fast velocity evaluation and adaptive mesh refinement to correct for particle distortion. For non-conservative discretization of differential operators (e.g.: DC–PSE, EGMLS), we intend to implement the new method of Quasi-Local Tree-based reconstruction (QLT) for communication efficient property preservation [8]. While other difficulties in the 2D realm

will undoubtedly require augmentations to our method, the results presented in this work display promise for the eventual adaption of LPM to the rotating shallow water equations on the sphere.

7. Acknowledgments. The authors thank Nathaniel Trask for several useful discussions. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR) Program and the Biological and Environmental Research (BER) Program under the Scientific Discovery through Advanced Computing (SciDAC) Partnership, and the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] R. A. Adams and J. J. Fournier, *Sobolev spaces*, vol. 140, Elsevier, 2003.
[2] J. M. Augenbaum, *A Lagrangian method for the shallow water equations based on a Voronoi mesh–one dimensional results*, Journal of Computational Physics, 53 (1984), pp. 240–265.
[3] L. Barba, A. Leonard, and C. Allen, *Advances in viscous vortex methods–meshless spatial adaption based on radial basis function interpolation*, International Journal for Numerical Methods in Fluids, 47 (2005), pp. 387–421.
[4] J. T. Beale and A. Majda, *High order accurate vortex methods with explicit velocity kernels*, Journal of Computational Physics, 58 (1985), pp. 188–208.
[5] A. Bihlo and S. MacLachlan, *Well-balanced mesh-based and meshless schemes for the shallow-water equations*, BIT Numerical Mathematics, 58 (2018), pp. 579–598.
[6] P. Bosler, L. Wang, C. Jablonowski, and R. Krasny, *A Lagrangian particle/panel method for the barotropic vorticity equations on a rotating sphere*, Fluid Dynamics Research, 46 (2014), p. 031406.
[7] P. A. Bosler, J. Kent, R. Krasny, and C. Jablonowski, *A Lagrangian particle method with remeshing for tracer transport on the sphere*, Journal of Computational Physics, 340 (2017), pp. 639–654.
[8] A. M. Bradley, P. A. Bosler, O. Guba, M. A. Taylor, and G. Barnett, *Communication-efficient property preservation in tracer transport*, SIAM Journal on Scientific Computing (submitted), (2018).
[9] A. J. Chorin and J. E. Marsden, *A mathematical introduction to fluid mechanics*, Springer, 3 ed., 1993.
[10] G.-H. Cottet and P. D. Koumoutsakos, *Vortex methods: theory and practice*, Cambridge university press, 2000.
[11] B. Cushman-Roisin, *Exact analytical solutions for elliptical vortices of the shallow-water equations*, Tellus A, 39 (1987), pp. 235–244.
[12] P. Degond and S. Mas-Gallic, *The weighted particle method for convection-diffusion equations. I. The case of an isotropic viscosity*, Mathematics of computation, 53 (1989), pp. 485–507.
[13] ———, *The weighted particle method for convection-diffusion equations. II. The anisotropic case*, Mathematics of computation, 53 (1989), pp. 509–525.
[14] O. Delestre, C. Lucas, P.-A. Ksinant, F. Darboux, C. Laguerre, T.-N.-T. Vo, F. James, and S. Cordier, *SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies*, International Journal for Numerical Methods in Fluids, 72 (2013), pp. 269–300.
[15] J. D. Eldredge, T. Colonius, and A. Leonard, *A vortex particle method for two-dimensional compressible flow*, Journal of Computational Physics, 179 (2002), pp. 371–399.
[16] J. D. Eldredge, A. Leonard, and T. Colonius, *A general deterministic treatment of derivatives in particle methods*, Journal of Computational Physics, 180 (2002), pp. 686–709.
[17] L. C. Evans, *Partial Differential Equations*, vol. 19 of Graduate Studies in Mathematics, AMS, Providence, Rhode Island, 1998.
[18] Y. Farjoun and B. Seibold, *An exactly conservative particle method for one dimensional scalar conservation laws*, Journal of Computational Physics, 228 (2009), pp. 5298–5315.
[19] T. Y. Hou and J. Lowengrub, *Convergence of the point vortex method for the 3-D Euler equations*, Communications on Pure and Applied Mathematics, 43 (1990), pp. 965–981.

[20] Y. Kimura and H. Okamoto, *Vortex motion on a sphere*, Journal of the Physical Society of Japan, 56 (1987), pp. 4203–4206.
[21] P. Koumoutsakos, *Multiscale flow simulations using particles*, Annu. Rev. Fluid Mech., 37 (2005), pp. 457–487.
[22] R. Krasny, *Desingularization of periodic vortex sheet roll-up*, Journal of Computational Physics, 65 (1986), pp. 292–313.
[23] ———, *A study of singularity formation in a vortex sheet by the point-vortex approximation*, Journal of Fluid Mechanics, 167 (1986), pp. 65–93.
[24] P. Lancaster and K. Salkauskas, *Surfaces generated by moving least squares methods*, Mathematics of computation, 37 (1981), pp. 141–158.
[25] P. H. Lauritzen, C. Jablonowski, M. A. Taylor, and R. D. Nair, *Numerical techniques for global atmospheric models*, vol. 80, Springer Science & Business Media, 2011.
[26] A. Leonard, *Computing three-dimensional incompressible flows with vortex elements*, Annual Review of Fluid Mechanics, 17 (1985), pp. 523–559.
[27] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, vol. 31, Cambridge university press, 2002.
[28] A. J. Majda and A. L. Bertozzi, *Vorticity and incompressible flow*, vol. 27, Cambridge University Press, 2002.
[29] R. Malek-Madani, *Physical oceanography: a mathematical introduction with MATLAB*, Crc Press, 2012.
[30] D. Mirzaei, R. Schaback, and M. Dehghan, *On generalized moving least squares and diffuse derivatives*, IMA Journal of Numerical Analysis, 32 (2012), pp. 983–1000.
[31] M. Nitsche and J. H. Strickland, *Extension of the gridless vortex method into the compressible flow regime**, Journal of Turbulence, 3 (2002), pp. 40–40.
[32] W. Pan, K. Kim, M. Perego, A. M. Tartakovsky, and M. L. Parks, *Modeling electrokinetic flows by consistent implicit incompressible smoothed particle hydrodynamics*, Journal of Computational Physics, 334 (2017), pp. 125–144.
[33] M. Perlman, *On the accuracy of vortex methods*, Journal of Computational Physics, 59 (1985), pp. 200–223.
[34] C. S. Peskin, *The immersed boundary method*, Acta numerica, 11 (2002), pp. 479–517.
[35] C. Rong-Jun and C. Yu-Min, *A meshless method for the compound KdV-Burgers equation*, Chinese Physics B, 20 (2011), p. 070206.
[36] J. Sampson, A. Easton, and M. Singh, *Moving boundary shallow water flow above parabolic bottom topography*, Anziam Journal, 47 (2006), pp. 373–387.
[37] B. Schrader, S. Reboux, and I. F. Sbalzarini, *Discretization correction of general integral PSE operators for particle methods*, Journal of Computational Physics, 229 (2010), pp. 4159–4182.
[38] H. Stommel, *The westward intensification of wind-driven ocean currents*, Eos, Transactions American Geophysical Union, 29 (1948), pp. 202–206.
[39] J. C. Strikwerda, *Finite difference schemes and partial differential equations*, 2004.
[40] W. C. Thacker, *Some exact solutions to the nonlinear shallow-water wave equations*, Journal of Fluid Mechanics, 107 (1981), pp. 499–508.
[41] K. S. Thorne and R. D. Blandford, *Modern Classical Physics: Optics, Fluids, Plasmas, Elasticity, Relativity, and Statistical Physics*, Princeton University Press, 2017.
[42] N. Trask, M. Maxey, and X. Hu, *Compact moving least squares: An optimization framework for generating high-order compact meshless discretizations*, Journal of Computational Physics, 326 (2016), pp. 596–611.
[43] ———, *A compatible high-order meshless method for the Stokes equations with applications to suspension flows*, Journal of Computational Physics, 355 (2018), pp. 310–326.
[44] N. Trask, M. Perego, and P. Bochev, *Mitigation of the self-force effect in unstructured PIC codes using Generalized Moving Least Squares*, Computers & Mathematics with Applications, (2018).
[45] G. K. Vallis, *Atmospheric and oceanic fluid dynamics*, Cambridge University Press, 2 ed., 2017.
[46] H. Wendland, *Scattered data approximation*, vol. 17, Cambridge university press, 2004.
[47] D. L. Williamson, J. Hack, R. Jakob, P. N. Swarztrauber, and J. Drake, *A standard test set for numerical approximations to the shallow water equations in spherical geometry*, tech. rep., Oak Ridge National Lab., TN (United States), 1991.
[48] Y. Xing and C.-W. Shu, *High order finite difference WENO schemes with the exact conservation property for the shallow water equations*, Journal of Computational Physics, 208 (2005), pp. 206–227.
[49] J.-M. Zokagoa and A. Soulaïmani, *A POD-based reduced-order model for free surface shallow water flows over real bathymetries for Monte-Carlo-type applications*, Computer Methods in Applied Mechanics and Engineering, 221 (2012), pp. 1–23.

REMOVING DEGENERATE FEATURES FROM ICE-SHEET MESHES

IAN A. BOGLE*, KAREN D. DEVINE†, MAURO PEREGO‡, SIVA RAJAMANICKAM§, AND GEORGE M. SLOTA¶

Abstract. In order to run climate simulations involving ice-sheets, it is important to be able to detect and remove features of the ice-sheet mesh that are problematic for the solvers that operate on those meshes. Because these degenerate features can appear over the course of many simulation steps, it is necessary to check for them after each step. We present a parallelizable BFS-based label-propagation approach that is efficient enough to be called at each step of an ice-sheet simulation, while still correctly identifying and removing all degenerate features of the ice-sheet mesh. We can find all degenerate features on a mesh with 13 million vertices in 1.39 seconds in serial on a single core. We also show that this approach can be generalized to finding all biconnected components of an undirected graph.

1. Introduction. Modeling Sea Level Rise (SLR) is an important objective of climate modeling. One of the biggest factors contributing to SLR is the melting of the ice caps [1]. In order to predict SLR accurately, we must be able to accurately model the changing of the ice-sheets in a computational environment. The most common way of representing three dimensional ice-sheets is to use meshes. For our purposes, we will look at these meshes like undirected graphs with a few more constraints than regular undirected graphs.

Over the course of an ice-sheet simulation, the mesh representing that ice-sheet changes, as the solvers that drive the simulation calculate how the ice melts and grows. Most of these changes do not cause any issues for the solvers that simulate these them, but there are circumstances that prevent solvers from being able to complete the simulation. These features, which we term "degenerate features" are discussed in Tuminaro et al. [8] and must be removed from the mesh in order to allow the solvers to continue their calculations.

Degenerate features of an ice mesh are very closely tied to which parts of the mesh are attached to the ground. We refer to vertices of the mesh that are in contact with the ground as "grounded" vertices. The situations that present a problem to the solvers are the cases where part of the ice mesh is floating in the water in such a way that it can move or rotate freely. The two cases of degenerate features that we are concerned with are floating islands, and hinged peninsulas. Floating islands are portions of the ice mesh that are not in contact with any grounded part of the mesh, and hinged peninsulas, or "hinges" are floating portions of the ice mesh that have only one point of contact with a grounded part of the mesh. A relevant concept from graph theory, articulation points, which are vertices that, when removed turn one connected graph into two separate connected graphs, could be useful in finding where hinged peninsulas are connected to some grounded portion of the mesh. This is the case, because the vertex that connects the peninsula to the rest of the mesh is an articulation point, since removing it would disconnect the hinged peninsula from the rest of the mesh, forming two separate graphs. From this insight, using an algorithm that can find every articulation point in the mesh seems like a natural part of a solution to this problem.

Finding all of the articulation points of the mesh, however, treats this problem as if our input is an undirected graph about which we can make no assumptions. In reality, the mesh structure of our input is very useful, in that it allows us to quickly make very good guesses about where articulation points may exist in the mesh. Even so, finding the articulation points does not completely solve the problem of identifying degenerate features.

*Rensselaer Polytechnic Institute, boglei@rpi.edu
†Sandia National Laboratories, kddevin@sandia.gov
‡Sandia National Laboratories, mperego@sandia.gov
§Sandia National Laboratories, srajama@sandia.gov
¶Rensselaer Polytechnic Institute, slotag@rpi.edu

Because degenerate features are caused by a lack of grounded vertices, we decided to design an approach that "spread" the grounding information through the mesh in such a way that degenerate features would become apparent. We used our guesses at which vertices in the mesh could be articulation points in order to restrict the potential propagation of grounding information to hinged peninsulas. This more tailored approach not only solves the problem, but our experimental results in figure 6.1 show that it is faster in serial than algorithms that find all the articulation points in a graph, and label the different parts of the graph that are separated by articulation points.

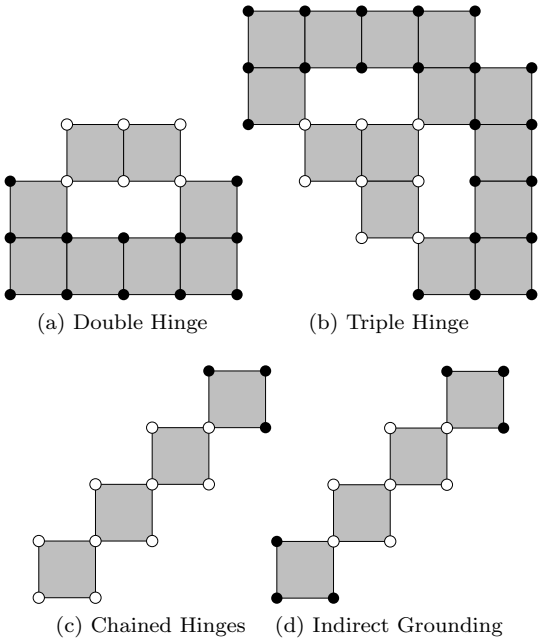


Fig. 1.1: Examples of mesh data from our dataset. Black nodes represent grounded vertices, white nodes represent floating vertices.

1.1. Our Contributions. We present an efficient serial algorithm for identifying degenerate features of ice meshes, and we show that our serial algorithm is faster than state-of-the-art biconnectivity algorithms. Additionally we show that the serial algorithm is amenable to a distributed memory implementation, and show how to generalize the algorithm to find biconnected components.

2. Background.

2.1. Previous Approach. The previous solution to this problem was implemented as a standalone MatLab code [8]. To remove floating islands, Tuminaro et al. used a Breadth-First Search (BFS) from every unvisited vertex in contact with the ground. Anything that remained unvisited after those BFSs has to be a floating island. To remove hinged floating peninsulas, they first colored the vertices of the mesh such that no neighboring vertices were the same color. Then, for each color, they removed vertices of that color and re-ran the ice island algorithm; hinged peninsulas would eventually become floating islands, because the hinge vertex would eventually be temporarily removed from the graph. This approach

generally had runtimes in the order of several minutes.

2.2. Biconnectivity Algorithms. The biconnectivity problem is concerned with finding components of the input graph which remain connected when any vertex is removed. Such components are referred to as biconnected components, while vertices that disconnect the graph are called articulation vertices, or articulation points.

Biconnectivity is a well studied problem, with a work optimal serial algorithm presented by Hopcroft and Tarjan [5]. This work optimal algorithm uses a Depth First Search (DFS) to identify biconnected components. DFS algorithms, however, are not easy to parallelize, as discussed by Tarjan and Vishkin [7]. Tarjan and Vishkin [7] present a parallel algorithm for finding biconnected components in a concurrent-read, concurrent-write parallel RAM model, where each processor has access to shared memory. This parallel algorithm reduces the problem of biconnectivity to the problem of connectivity on an auxiliary graph. The auxiliary graph is able to be constructed without using a DFS, so Tarjan and Vishkin were able to present an efficiently parallelizable algorithm in shared memory.

We used two general biconnectivity algorithms for various tests against our approach [6]. These algorithms use simple graph operations such as BFS or color propagation in order to decompose a graph into its biconnected components. The BFS-based algorithm does an initial BFS sweep, and then subsequents BFS sweeps check to see if children of certain nodes can reach other nodes on their parents' level. If so, the parent is not an articulation point. The coloring version also does an initial BFS, but it uses color propagation rules to prevent certain labels from being passed through articulation points. Originally, we were planning on using these algorithms to solve this problem. We planned on running a biconnectivity algorithm, and then doing a BFS-like check to make sure that the separate biconnected components were all attached to the ground. These two algorithms are not work optimal in the general case, but they achieve speedup due to the fact that they are comprised of efficiently parallelizable subroutines. However, it is likely that these shared-memory parallel algorithms are not efficiently parallelizable in a distributed memory context.

3. Ice-sheet feature detection. Algorithm 12 shows a high-level overview of our approach. As shown, our algorithms generally require grounding information about the nodes, and some guess as to which nodes are articulation points. It is important to node that this heuristic is only needed for speed, and not correctness. However, the correctness of the heuristic is important, because the propagation rules assume that the set of potential articulation points contains all actual articulation points.

Algorithm 12 Our general approach for both biconnectivity and ice sheets

```
1: procedure PROP-ALG(G=(V,E),grounding_info)
2:     labels ← ∅
3:     Compute articulation heuristic using G
4:     Propagate labels
5:     while Propagation needs fixed do
6:         Fix propagation, and repropagate
7:     end while
8:     Return labels
9: end procedure
```

Our proposed algorithm has three inputs: A set of vertices that contain potential articulation points , where the set of actual articulation points is a subset of this set; Grounding information for each vertex; and the graph representation of the mesh. Our labels

in this approach hold four vertex IDs: two representing grounded vertices, and the other two keep track of which vertex sent which ID. We initialize a set of labels for all vertices from the initial grounding information, and start propagating the labels from each grounded node. We use two frontiers, $frontier$, and $art\_frontier$, and in a BFS-like manner, we only examine a node's neighbors if that node has gained a grounded vertex ID in its label since we last saw it on the frontier. We use $art\_frontier$ for potential articulation points, and $frontier$ for regular nodes, in order to allow the potential articulation points' labels to become more full before they communicate labels to their neighbors. In short, we do this because we only allow potential articulation points to communicate one vertex ID, and if they communicate too soon, they could cause incomplete propagation (see section 3.4). After this propagation is complete, we check to see if there are any signs of incomplete propagation, and we fix the incomplete propagation if necessary. Then, any nodes that have two grounded vertex IDs in their label are nodes we keep, and any nodes with less than two grounded vertex IDs are nodes we remove.

3.1. Mesh Articulation Point Heuristic. When operating on an ice-sheet mesh, it is fairly easy, given the elements of the mesh, to compute which edges in the mesh form the boundary between the ice and the water. The edges that only appear once in the list of elements are the ones that are on the boundary, because there is no element on the other side of that edge. Given the boundary edges, we can determine which nodes are potentially articulation points by looking at how many boundary edges are incident to a given node. If a node has two or less incident boundary edges, then it cannot be an articulation point, because those nodes are on at most one boundary-edge cycle. If the node has more than two incident boundary edges, then it is potentially an articulation point. (show a figure of this process here)

3.2. Label Propagation Rules. Algorithm 13 shows the logic used to pass labels between neighboring vertices. We ensure that all potential articulation points send only one label to each neighbor, because all articulation points can only contribute one point of contact to the ground to any neighbor. The rest of the nodes can pass as many labels as they want, since they are guaranteed not to be articulation points. It should be noted that the "giving" of labels mentioned in the algorithm simply checks to make sure that labels consist of the vertex IDs of unique, grounded nodes.

Algorithm 13 Function for passing labels to neighbors

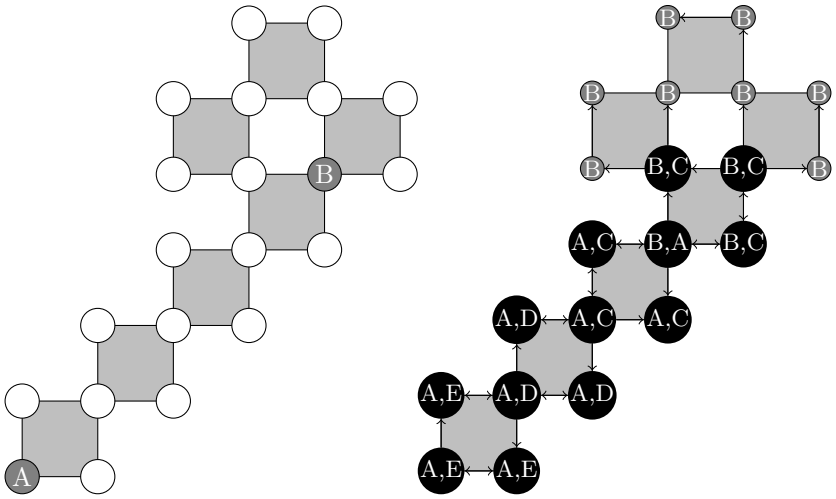
```
1:  procedure GIVE-LABELS(curr_node, neighbor)
2:      if curr_node ∈ Potential_Articulation_Points then
3:          if curr_node hasn't sent anything to neighbor before then
4:              if curr_node has two grounded vertex IDs then
5:                  curr_node gives neighbor its own vertex ID as a grounded vertex
6:              else
7:                  curr_node gives its only grounded vertex ID to neighbor
8:              end if
9:          end if
10:     else
11:         curr_node gives neighbor any grounded vertex IDs neighbor doesn't have
12:     end if
13: end procedure
```

3.3. Two Frontiers Approach. As shown in Algorithm 14, the two frontiers separate the potential articulation points from the other nodes while propagating. This is done so that the potential articulation points have a better chance to be in their final label-state, and pass on vertex IDs that will not cause incomplete propagation. This approach is good enough for all the real ice-sheet data we have dealt with, but it is possible to break it with a synthetic example.

Algorithm 14 BFS-based label propagation algorithm

```
1:  procedure BFS-Prop(frontier,labels,G=(V,E))
2:      art_frontier ← ∅
3:      while !frontier.empty() do
4:          curr_node ← frontier.pop()
5:          for all neighbors n of curr_node do
6:              Give-Labels(curr_node,n)
7:              if n's label changed then
8:                  if n ∈ Potential_Articulation_Points then
9:                      art_frontier.push(n)
10:                 else
11:                     frontier.push(n)
12:                 end if
13:             end if
14:         end for
15:         if frontier.empty() then
16:             swap(frontier,art_frontier)
17:         end if
18:     end while
19: end procedure
```

3.4. Incomplete Propagation. Incomplete propagation is a complex situation that is prominent in the generalized case, but almost nonexistent in our ice-sheet data. It is best illustrated by figure 3.1. To fix this issue, we check for the signs of incomplete propagation, which are potential articulation points that end up with two grounded vertex IDs in their label, but did not send their own vertex ID to their neighbors (The vertex that neighbors B in figure 3.1 is an example of this state). We add all such nodes to a frontier, clear out any labels that have only one grounded vertex ID in them, and restart propagation from this new frontier. Algorithm 15 shows this addition to the BFS-Prop algorithm.



(a) Before propagation (b) After propagation

Fig. 3.1: Because B's vertex ID reaches two potential articulation points before A's label can, we have incomplete propagation.

Algorithm 15 Driver for BFS-Prop, fixes incomplete propagation

```
1:  procedure BFS-Prop-Driver(labels, G=(V,E))
2:      frontier ←vertex IDs of half labeled vertices
3:      BFS-Prop(frontier,labels,G)
4:      while true do
5:          for all v ∈ V do
6:              if v's label is full then
7:                  for all neighbors n of v do
8:                      if n's label is half-full and v's ID is not present in it then
9:                          frontier.push(v)
10:                     end if
11:                 end for
12:             end if
13:         end for
14:         if frontier.empty() then
15:             break
16:         end if
17:         for all v ∈ V do
18:             if v's label is half full then
19:                 clear v's label
20:             end if
21:         end for
22:         BFS-Prop(frontier,labels,G)
23:     end while
24: end procedure
```

4. Generalization to Biconnnected Component Detection. In generalizing this tailored algorithm to find biconnected components on general undirected graphs, we needed to find a suitable heuristic for finding potential articulation points, and we needed to somehow manufacture grounding information.

4.1. LCA heuristic. LCA, or Lowest Common Ancestor, is a measure that makes sense only in terms of trees. Thus, given a spanning tree T, the LCA of two vertices is the lowest node in T that has both vertices as descendants. Given two vertices in a spanning tree, a simple serial algorithm would be to follow the parents of both nodes until the traversals meet at the LCA vertex.

The generalized heuristic that we settled on was found in [3]. The idea is to do a BFS, find the Lowest Common Ancestor of each non-tree edge, and then note which nodes were not visited in the traversals to find the LCAs. This heuristic, proved by [3], guarantees that all articulation points will be contained in the set of vertices it returns. However, in practice we found that it generally returned many more vertices than the Mesh Heuristic presented in section 3.1.

4.2. Manufacturing Ground. The straightforward choice of which vertices to ground initially is to ground any two neighboring vertices. This choice is advantageous because two neighboring vertices must be in the same BCC, so we cannot be grounding vertices in two separate biconnected components. After running Algorithm 15, any vertices that have full labels are in the same biconnected component, and any vertices that are full, but neighbor half-full or empty vertices are articulation points. We clear out all the half labels from the last propagation in order to prepare for the next one. Then, we ground the neighbor of an articulation point to find the next BCC, and continue doing so until all vertices in the graph have full labels.

4.3. Final BCC Decomposition. A minor tweak to Algorithm 13 is needed in order to easily label the Biconnected Components of the input graph. During each call of Algorithm 15, we also pass along the name of that biconnected component. Any node that ends up with two labels will retain that name, while all other nodes will have their labels reset. Algorithm 16 shows the full algorithm.

Algorithm 16 Generalized Driver to find Biconnected Components

```
1: procedure BCC-BFS-Prop(labels, G=(V,E))
2:     art_queue ← ∅
3:     frontier ← ∅
4:     while not every label is full do
5:         while frontier.empty() do
6:             if !art_queue.empty() then
7:                 ground a neighbor of art_queue.front()
8:                 push both nodes onto frontier
9:                 if art_queue.front() has no empty neighbors then
10:                    art_queue.pop()
11:                end if
12:            else
13:                ground two empty neighbors
14:                push both nodes onto frontier
15:            end if
16:        end while
17:        BFS-Prop-Driver(labels, G=(V,E))          ▷ Using modified labels
18:        for all v ∈ V do
19:            if v's label is full then
20:                for all neighbors n of v do
21:                    if n's label is not full then
22:                        art_queue.push(v)
23:                        break
24:                    end if
25:                end for
26:            end if
27:        end for
28:        for all v ∈ V do
29:            if v's label is not full then
30:                clear v's label
31:            end if
32:        end for
33:    end while
34: end procedure
```

5. Distributed Memory Implementation.

5.1. Ice Sheet Version. Because the ice sheet mesh that we will be operating on exists in distributed memory, we needed to implement this algorithm in a distributed memory environment as well. The algorithm adapts surprisingly well to a distributed memory environment, in that it can maintain its "push coloring" label propagation strategy, without causing any ill effects. In our case, we are assuming that the graph distribution is provided by the application, but that we need to ghost the neighboring vertices along the partition boundaries. Essentially, we treat each processor's part of the graph as its own instance of a smaller problem. Once propagation stops, the labels on the ghosted vertices are communicated, and made consistent with their owned versions. Then, if the communication significantly changed any labels for any processor, the propagation continues. Synchronization is required between the initial call to BFS-Prop and the subsequent calls to

BFS-Prop that fix incomplete propagation.

We heavily used existing objects from the Tpetra [2] package of Trilinos [4] in order to implement this strategy. We used Tpetra Maps to initially set up the layouts of the owned and ghosted vertices, and used those Maps to create a Tpetra FEMultiVector. The FEMultiVector came with functionality to communicate accross processor boundaries built-in, all we had to do to implement our label propagation scheme was overload the += operator for the type that represented our labels. In order to use the FEMultiVector's communication capabilities, we had to call beginFill() before we started our per-processor label propagation, and endFill() to communicate the ghosted labels back to their owned versions. However, we also needed to communicate the owned labels back to the ghosted versions, because at this point they could be inconsistent, so we called doSourceToTarget() with Tpetra::ADD as one of the arguments, to ensure our label-swapping logic was used.

5.2. Generalized Version. The generalized version of this algorithm is slightly less straightforward to implement in a distributed memory environment, because of the heuristic we used. The label propagation phase is just a minor variation of the Ice Sheet version. Before we run the LCA Heuristic, we run a an implementation of a distributed BFS.

5.2.1. LCA Heuristic. The distributed memory implementation of this heuristic centers on the use of queues. Each processor has a queue of entries corresponding to an LCA traversal. Each individual entry contains the two active vertices, the level of both vertices, and the numbers of the tasks that own each vertex. While a processor works on the traversals, eventually the travesal will end, or need to be passed to a different processor. Once all processors run out of work to do locally, they communicate the entries they cannot advance to a processor that can. Then, each processor may have new entries to work on locally, and the cycle repeats. This cycle ends once all processors have nothing to communicate, and no more local work to do. As the entries advance, we also mark the edges that we've visited, and both the LCA vertices and the endpoints of unvisited edges become potential articulation points.

5.2.2. BCC BFS-Prop Driver. As in the generalized serial version, the distributed version essentially runs the distributed ice-sheet propagation function multiple times until each biconnected component of the input graph is found. In order to increase parallelization, we attempt to make the initially grounded neighbors be nodes that are split across a processor boundary. Like the serial version, after the first propagation finishes, we can identify actual articulation points that are incident to the biconnected component we just found by looking for vertices that have two labels, but neighbor vertices that only have one label. After this identification takes place, we clear out the half labels on all the processors. Then, we can safely ground one neighbor of each articulation point in order to find multiple biconnected components in parallel. If we run out of articulation points before we finish labeling the whole graph, we simply ground two neighbors again, favoring neighbors that are split across a processor boundary. We keep doing propagations until each processor's vertices all have two labels. In order to keep the names of each biconnected component unique, we have each processor keep a count of how many biconnected components they have found, and the name of the next biconnected component is found by $bcc\_count * \#processors + processor\_ID$.

6. Results.

6.1. Experimental Setup. The runtimes we present were gathered on Blake, a Hybrid Advanced Architecture Platform at Sandia. Blake has 40 nodes with Dual-socket Intel Xeon Platinum processors. Each of these nodes has 12 cores, and Blake has an Intel Omnipath interconnect.

6.2. Ice-sheet Results. The largest input ice-sheet mesh had 13 million vertices, and about 13 million edges, and was obtained from the team running climate simulations. Table 6.1 shows the serial runtime of our algorithm against the runtimes of serial biconnectivity algorithms. Our algorithm achieves an 2.39x speedup over the bcc-bfs algorithm on the largest input, and a much larger speedup over the bcc-color, 161x on the second largest input. We show this result to motivate our attempt at generalization, this algorithm is faster in serial than BCC algorithms, and more amenable to a distributed memory implementation.

Runtimes with OMP_NUM_THREADS=1			
Ice-sheet Resolution	BCC-BFS	BCC-Color	Tailored Approach
16km	0.0163 (s)	0.0841 (s)	0.0046 (s)
8km	0.0483 (s)	0.7728 (s)	0.0196 (s)
4km	0.1912 (s)	7.6713 (s)	0.0834 (s)
2km	0.7199 (s)	54.821 (s)	0.3395 (s)
1km	3.3271 (s)	—— (s)	1.3904 (s)

Fig. 6.1: Runtimes of the serial ice-sheet algorithm vs. serial biconnectivity algorithms on blake

6.3. Generalization Results. The graphs used as input to the distributed BCC each had two biconnected components, and the number of edges equals two times the number of vertices. These graphs were generated by first constructing two cycles, and independently connecting their respective nodes with a random chance. The number of random edges was capped at the number of vertices, to keep the graphs relatively sparse. The table 6.2 shows the runtime of BCC-BFS-Prop part the algorithm on different numbers of cores, with one MPI rank per core. The runtimes for the 4 million vertex graph initially jump up from the serial runtime, most likely due to higher overhead in communication, while not gaining enough parallelism with small numbers of cores. The maximum speedup achieved on the 4 million vertex graph is 33x, with 528 processors. The number of cores for the 40 million vertex graph was chosen due to the fact that a multiple of 12 MPI processes gave the best speedup for the 4 million vertex graph, and that each processor on Blake has 12 cores. The maximum speedup achieved for the 40 million vertex graph is 15x. Much less data was gathered for the 40 million edge graph because of how much time it took to read the file and distribute the graph. Additionally, no times are reported for the distributed LCA algorithm because we have not yet implemented the LCA heuristic in distributed memory.

Distributed BCC-BFS-Prop Runtimes		
# processors	4 Million Vertex Graph	40 Million Vertex Graph
1	95.436(s)	–
2	101.088(s)	–
4	116.19(s)	–
8	91.39(s)	–
16	58.53(s)	–
32	32.28(s)	–
48	26.568(s)	224.49(s)
64	17.783(s)	–
128	10.442(s)	–
256	6.011(s)	–
512	3.441(s)	–
528	2.8647(s)	27.957(s)
1024	3.417(s)	–
1056	3.1772	14.686(s)

Fig. 6.2: Runtimes of the distributed BCC algorithm on blake

7. Conclusions and Future Work. We have proposed an algorithm for identifying degenerate features in ice-sheet meshes. We showed that a serial implementation ran 2x faster than a serial Biconnectivity algorithm, and that the algorithm lends itself to a straightforward distributed memory implementation. Additionally, we showed how the version of the algorithm that is tailored towards ice-sheet meshes can be generalized to find the Biconnected Components of any undirected graph.

Future work related to this algorithm could take many directions. The label propagation function may be parallelizable in shared memory, so a hybrid distributed and shared memory implementation may be possible. There are optimizations that can be made to the generalized LCA heuristic, and it could be that grounding two neighboring vertices is not the most efficient way to start the general BCC algorithm. Finally, we have not explored what using three labels for each node would mean in the general case, but it could lead to a distributed memory triconnectivity algorithm.

REFERENCES

[1] R. B. Alley, P. U. Clark, P. Huybrechts, and I. Joughin, *Ice-sheet and sea-level changes*, Science, 310 (2005), pp. 456–460.
[2] C. Baker and M. Heroux, *Tpetra, and the use of generic programming in scientific computing*, Scientific Programming, 20 (2012), pp. 115–128.
[3] M. Chaitanya and K. Kothapalli, *Efficient multicore algorithms for identifying biconnected components*, Int. J. Networking and Computing, 6 (2016), pp. 87–106.
[4] M. A. Heroux and J. M. Willenbring, *A new overview of the trilinos project*, Scientific Programming, 20 (2012), pp. 83–88.
[5] J. Hopcroft and R. Tarjan, *Algorithm 447: Efficient algorithms for graph manipulation*, Commun. ACM, 16 (1973), pp. 372–378.
[6] G. M. Slota and K. Madduri, *Simple parallel biconnectivity algorithms for multicore platforms*, in International Conference on High Performance Computing (HiPC), 2014.
[7] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM Journal on Computing, 14 (1985), pp. 862–874.
[8] R. Tuminaro, M. Perego, I. Tezaur, A. Salinger, and S. Price, *A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling*, SIAM J. Sci. Comput., 38 (2016), pp. C504–C532.

CURVATURE BASED ANALYSIS TO IDENTIFY AND CATEGORIZE TRAJECTORY SUBSEGMENTS

PAUL T. SCHRUM, JR.*, MARK D. RINTOUL†, AND BENJAMIN D. NEWTON‡

Abstract. Since the attacks carried out against the United States on September 11, 2001, which involved the commandeering of commercial aircraft, interest has increased in performing trajectory analysis of vehicle types not constrained by roadways or railways, i.e., aircraft and watercraft. Anomalous trajectories need to be automatically identified along with other trajectories of interest to flag them for further investigation. There is also interest in analyzing trajectories without a focus on anomaly detection. Various approaches to analyzing these trajectories have been undertaken with useful results to date. In this research, we seek to augment trajectory analysis by carrying out analysis of the trajectory curvature along with other parameters, including distance and total deflection (change in direction). At each point triplet in the ordered sequence of points, these parameters are computed. Adjacent point triplets with similar values are grouped together to form a higher level of semantic categorization. These categorizations are then analyzed to form a yet higher level of categorization which has more specific semantic meaning. This top level of categorization is then summarized for all trajectories under study, allowing for fast identification of trajectories with various semantic characteristics.

1. Introduction. Whenever an object moves, its path is called a trajectory. For vehicles, these trajectories may be recorded over time by occasionally sampling the location in a certain coordinate reference frame. The resulting sequence of time-stamped position samples constitutes the "trajectories" we study in this research.

Of these vehicles, aircraft and water-borne vehicles are generally not constrained by predetermined pathways, as is the case for road-borne and rail-borne vehicles. Thus aircraft and watercraft have a large number of possible variations in path from one point to another. Furthermore, aircraft and watercraft have been used as transportation for threat agents specifically because of this greater freedom of course coverage, and in some cases the vehicles themselves have been used as the destructive weapon. It would be advantageous if unusual trajectories could be identified by analyzing their geometry as described by the sequence of sampled points. Because of these and other motivations, several researchers have undertaken to analyze and characterize trajectories from the recorded point sequences.

One general approach to trajectory analysis involves computation of the normalized parameter space or normalized feature space. This approach is used by [1], and [3]. [1] performs an image parsing technique, Recursive Multi-frequency Segmentation, cross applied to the speed parameter of bird flights to detect speed-based segments. [3] computes whole-trajectory aggregate values including total distance, and convex hull aspect ratio.

The method we have explored in this research takes a sequential point-by-point approach, looking at the geometry of the trajectory at each point and its nearest previous and next neighbors along the ordered sequence. This approach yields information at a detailed level which is then aggregated into groups of similar properties until the entire trajectory has been partitioned into meaningful subsegments. The 'meaningful subsegments' are referred to as Level 1 Categorizations, which are the primary result yielded by the method. The Level 1 Categorizations are reduced to data formats which are well suited to established data mining techniques and may be used for various purposes.

We present the method, which we call "The Curvature Parsing Method" (CPM), demonstrate its capability, and assess its strengths and weaknesses.

Although the method is intended to be generalizable to maritime trajectories, only

*North Carolina State University, ptschrum@ncsu.edu
†Sandia National Laboratories, mdrinto@sandia.gov
‡Sandia National Laboratories, bdnewto@sandia.gov

aviation data was used in the development of these algorithms, so further discussion focuses mainly on aviation trajectories.

2. Methods. The core of the Curvature Parsing Method is shown below. The subsequent discussion on the method follows this pseudocode as an outline.

1. Load batch of trajectory data into memory.
2. For each trajectory:
   (a) Compute local point geometry, including point triple curvature.
   (b) Perform Level 3 Processing: Categorize Each Individual Point.
   (c) Level 2 Processing: Group adjacent similar Level 3 Nodes into subsequences and categorize.
   (d) Level 1 Processing: Aggregate Level 2 Nodes into subsequences based on sequence patterns and categorize.
   (e) Generate intermediate data outputs as requested for each trajectory.
3. Generate report summarizing results for each Level 1 Node for the batch.

The items in this listing serve as an outline for the Methods section.

Optional data outputs are
1. a kml file for visualization using Google Earth (.kml) [1] ,
2. a plot of the parse tree graph (.png),
3. a detailed arc data report including derived geometry parameters (.csv),
4. a statistical summary report of Level 1 categorizations (.csv),
5. a Level 1 hash bin report grouping all trajectories by a hash string of the Level 1 categorization. (.csv)

The details of these output types are provided in Section 2.3.4. Figures 2.1 and 2.2 together illustrate two of these outputs for a single trajectory. Figure 2.1 shows a kml plot of a flight which begins in the cruise phase over Lake Erie and ends with a landing at Ottawa.

The parse tree graph of Figure 2.2 illustrates important features of the process, detailed below, including the way in which points constitute the leaves of the parse tree, the entire trajectory is the root of the tree, and Level 1 and 2 categorizations are between the root and the leaves.

2.1. Trajectory Data Source. Trajectory data used in the development of this project originates with Aircraft Situation Display to Industry (ASDI), a data feed of aircraft trajectories in United States and Canadian airspace provided by the Federal Aviation Administration (FAA). ASDI data includes "aircraft scheduling, routing, and positional information." [6] Our purposes only require use of the temporal positional information and the aircraft identity to distinguish individual trajectories.

The FAA obtains ASDI data from radar stations across North America, which gather positional information as well as information squawked by the aircraft such as identity and altitude. The data has not been optimized to eliminate anomalies. Specifically, the data sometimes contains points which are the result of two radars reporting the position of the same aircraft at slightly different positions and different time stamps, resulting in a zigzag pattern in occasional portions of the trajectory. We refer to this kind of data anomaly as radar jitter. Further, there are rare data anomalies in which two sequentially adjacent points have identical coordinates and time stamps. We refer to these as double-stamps. Both kinds

[1] It is also possible to output all kml files into subdirectories named for Level 1 hashes.

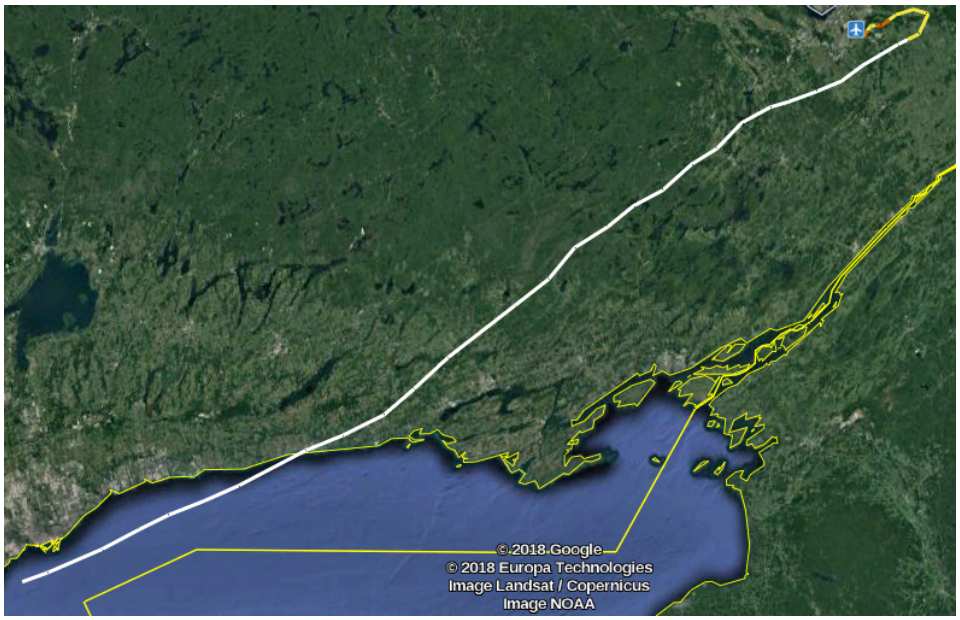


Fig. 2.1: Partial Trajectory of Flight A3A458 in which sampling begins mid-flight over Lake Erie and ends with landing in Ottawa. White segments are straight; Yellow segments are left turns; Orange segments are right turns.

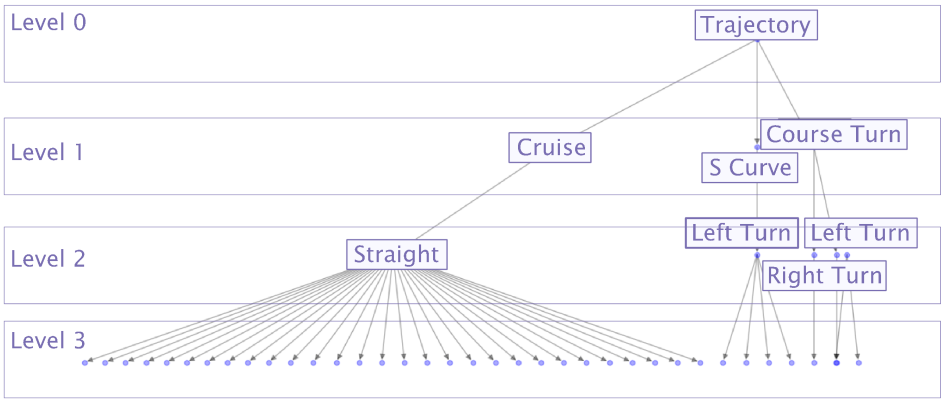


Fig. 2.2: The Parse Tree Graph of Flight A3A458, depicted in Fig.2.1.

of anomalies cause problems for the algorithm. Thus those points are not removed from the dataset, but the categorizer ignores them by continuing the previous categorization across the anomalies.

Rintoul et al., in working on Sandia National Lab's PANTHER project, call for "a more thorough analysis of the information content in the different features" to be carried out, and an "examination of more efficient ways to break up the trajectories into segments to find smaller features". ([4], p. 72) The current research is part of that follow on work. The software

developed for the research being presented here is an exploratory extension of Tracktable. The PANTHER report states, "Tracktable is an open source library which contains a core set of functionality for ingesting, processing, plotting, and analyzing trajectories." ([4], p. 39) The software implementation of the current research depends on Tracktable's Application Programming Interface (API) for data access and some computational geometry functionality. The desired outcome is a way to identify meaningful trajectory subsegments in partial support of a subset of PANTHER's stated goals:

1. finding trajectories that exhibit a behavior of interest without regard to translation, rotation or scale,
2. dividing trajectories into specific clusters,
3. finding trajectories that are outliers with respect to a given set of trajectories, and
4. performing analysis prelimanry to classifing trajectories using unsupervised learning techniques.

([4], p 18)

In some cases the categorizations performed in the current research depend on trajectory subsegment absolute length, which makes the analysis no longer invariant under scale transforms.

2.2. Compute Local Point Geometry. The basis of this algorithm is the notion that any three points define a portion of a circle. Figure 2.3 shows how three coplanar points uniquely define an arc segment regardless of the offset of the middle point.

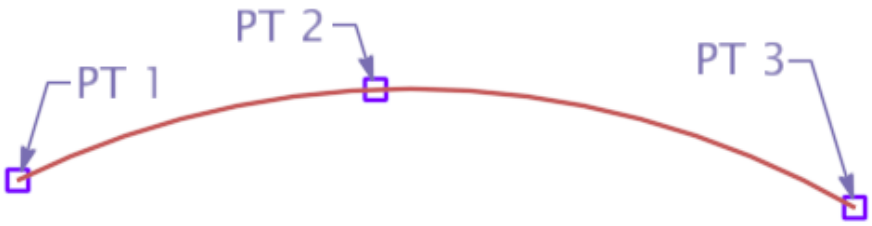


Fig. 2.3: Three points defining a portion of a circle. The interior solution is shown. Exterior solutions, the arcs going the long way around the are not used in the present algorithm.

Although the altitude input parameter is available for most trajectories, this algorithm does not use it, so all points are considered coplanar for the purpose of analysis. We are also developing an altitude based analysis in parallel with this research. At some point these two types of analyses may be merged for greater insight into aviation trajectories.

For each sequential point triplet of the given trajectory, the arc segment defined by point subsequence n-1, n, and n+1 is computed and the resulting computed information (hereafter referred to as 'curve data') is associated with point n. Hereafter, the midpoint of the point triplet, n, is referred to as the keypoint because in the data model, the arc segment parameters for arc n are stored with point n The first and last points of the sequence cannot have an associated arc, having only a single neighbor.

The terms 'trajectory' and 'alignment' are related but different. In the following discussion, 'trajectory' refers to the historical path of a physical object represented by a point moving through time; 'alignment' refers to a static, mathematically defined path independent of time. To clarify this distinction, roads are built along alignments, but cars trace out trajectories.

In fact, the disciplines of railway and roadway design, along with surveying, also concern themselves with determining the mathematical definition of smooth alignments from sampled data points. For this reason, some of the theory from those disciplines may be adapted for our purposes.

Concepts, symbols, and terminology are taken from [2], which is used extensively by surveyors and civil engineers for alignment definitions. Figure 2.4, adapted from [2] Figure 9 and accompanying discussion (pp 64-67), depicts the engineering symbology and definitions for the arc segment used to fillet two straight line segments. All geometry is flattened to eliminate the Z dimension. Further, the notion of Length in the following discussion only considers the 2D length of a path in the plane, and does not include any Z component. The parameters shown here are the curve Radius, R, and the curve deflection, Δ. PC is Point of Curvature, shared by the incoming tangent and the curve, PT, Point of Tangency, shared by the curve and the outgoing tangent, PI, the Point of Intersection of the two tangents without the curve, and CC, the Curve Center.

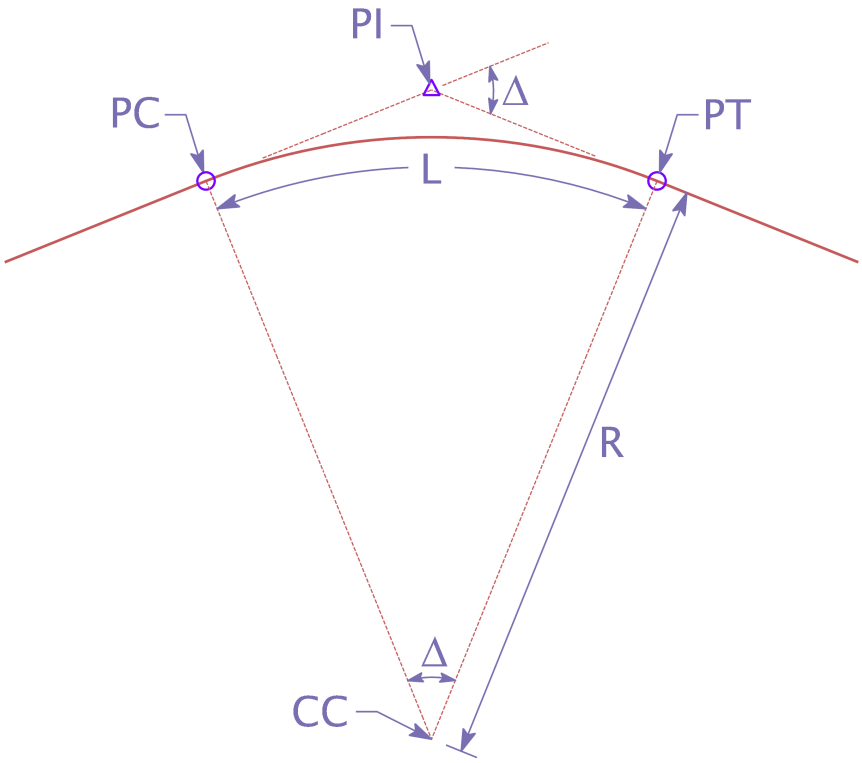


Fig. 2.4: Basic arc segment elements using Civil Engineering terminology. CC is the Curve Center. PI is the Point of Intersection of the tangential line segments at the begin and end of the curve. PC is "Point of Curvature", which is the point where the alignment transitions from the line segment to the arc segment. PT is "Point of Tangency", which is the point where the alignment transitions from the arc segment to a different line segment. L is the length along the arc segment. Δ, Deflection, is the total change of heading of the arc segment from the PC to the PT.

Curvature, also referred to as Degree of Curve (Dc), may be visualized as the degrees of heading change over one unit length along the trajectory or alignment in which the length of the arc changing by one degree is the same unit length for radius. Roadway design engineers in the United States use a length basis of 100 feet for Degree of Curve. For our research we work with a 1 mile length Degree of Curve basis. Figure 2.4 illustrates this concept. For the case where total curve length, L, equals 1 basis unit, then the total deflection, Δ, equals the curvature of the curve in degrees of change of heading per basis unit of length.

Several observations should be noted regarding Figure 2.4. The elements and equations for the engineering definition of an arc fillet are constrained by the requirement that all elements be continuous in heading. Curvature is the first derivative of heading. When a civil engineering alignment is used without Euler Spiral Segments (as depicted in Figure 2.4), there is a curvature discontinuity at the PC and PT, but heading is continuous. Following such an alignment precisely is physically impossible for real world vehicles due to conservation of momentum, so there are always easement curves at the beginning and end of a filleting curve which serve to ease from zero curvature to non-zero curvature and back. Civil engineers use Euler Spiral Segments to accomplish this easement, but easement curves are not being considered for this trajectory analysis.

Since curvature is the first derivative of heading, plots of heading versus length may conveniently be collocated on the same graph as curvature versus length. Figure 2.5, below, depicts such a simultaneous plot.

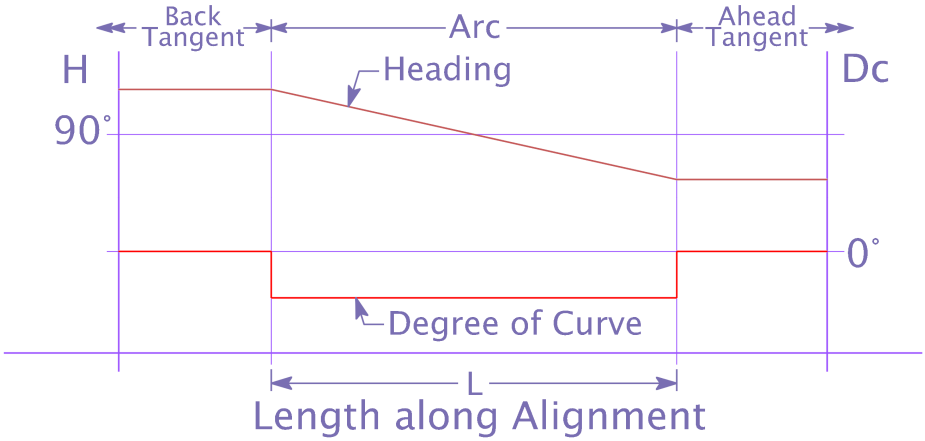


Fig. 2.5: Heading versus Length Along (top line) and Degree of Curve versus Length Along Alignment (bottom line). This collocated plot illustrates the first derivative relationship between Heading and Curvature. This plot matches the arc segment shown in Figure 2.4.

2.2.1. Limitations of Sampled Data. The plot in Figure 2.5, heading versus length combined with curvature versus length, is idealized for a proscribed engineering alignment such as the one shown in Figure 2.4. Sample-based trajectories have limitations due to sample noise and sampling rate. Regarding sample noise, if an aircraft is following a perfectly straight geodesic (also known as a Great Circle Route, the spherical equivalent of a straight line), noise will introduce errors into the reported position at each sample, leading to each

point being slightly off the actual trajectory, and the curvature, which would be zero for a straight line, will have values near zero on both the positive side and the negative side, averaging to near zero over multiple sequential samples.

Our dataset consists of points sampled about every 60 seconds in most cases. Since the sampling rate is not continuous, certain turning events may be skipped over or under sampled, resulting in ambiguous or unresolvable features. This phenomenon is conceptually related to the Nyquist-Shannon Sampling Theorem [5].

The sampling rate is determined in the time domain, but the comparison basis for sampling rate is in the length domain. So a slower aircraft has a shorter physical distance between samples (more samples per hundred miles) than a fast moving one, and this has a higher sample resolution even though both have the same number of samples per minute.

A related problem is partial aliasing of point triplets at arc boundaries. Specifically, if a given point is near the beginning or end of a curve, one of its neighbor points lies on the adjacent tangent, and one lies on the curve. The result is that the key point (the middle point of the triplet) shows a transitioning curvature value even though that point is on the tangent of the alignment. This is illustrated in Figure 2.6

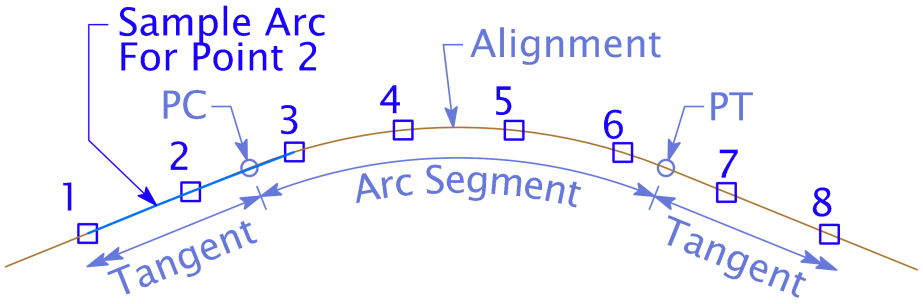


Fig. 2.6: Sample points along an alignment showing aliasing where some point triplets span an end of the arc segment

Figure 2.6 shows a contrived example of of sample points along a perfect alignment with no sample noise. The distance between samples is 6.0 miles. The sample arc associated with point 2 is highlighted in blue. Sample arcs 2, 3, 6, and 7 overlap the end points of the true curve, so their curve values are aliased between an arc and a tangent. Sample arcs 4 and 5 (not shown as arcs, but as points) fall completely on the true arc, so their curvatures will be the same as the true arc.

Table 2.1 illustrates the same effect numerically.

In addition to end-aliasing, certain curves of short length may be skipped by sample points altogether, while the triple-point curvature data yields curvature and Δ values which differ considerably from the actual value. In the extreme case, the curve length is so short compared to the sample rate that sample points fall before and after the arc segment, but none fall directly on it, so aliasing occurs at all points involved with the curve.

Another caveat of basing the analysis on curve segments of sampled data points is that neither chord deflection δ_{Chord} or total arc deflection Δ_{Sample} represent the true change of heading from Point 1 to Point 3 when curves are under sampled. In the categorizations process, where change of heading is used for categorization criteria, it is based on δ_{Chord} because the estimation error is much less.

Figure 2.7 illustrates the definitions of these terms. The alignment being sampled is the

Arc Segment	R	Dc	Δ_{Sample}
1-2-3	32.57 mi	1.76°	2.11°
2-3-4	3.59 mi	15.97°	19.17°
4-5-6	3.00 mi	19.06°	22.89°
True Curve	3.00 mi	19.06°	43.98°

Table 2.1: Transition Values for Curvature (Dc) for point triplets spanning the beginning of a curve. R is sample curve radius. Δ_{Sample} is sample curve deflection.

same alignment as shown in 2.4 and 2.5, although the sample points are sparser. Δ_{Sample} is the deflection of the arc generated from the point triplet, 1-2-3, shown in blue. δ_{Chord} is the deflection of the straight line segments 1-2 and 2-3. This figure also demonstrates the effect of the sampled arc curvature and radius being substantially different from the true curve when only one sample point falls on the arc. We believe this effect impacts the subsequent categorization processing, but the details have not been confirmed and should be investigated further.

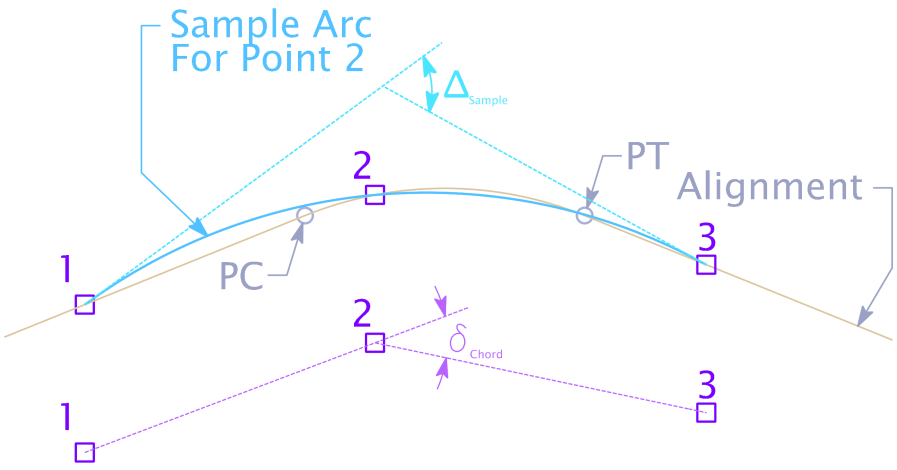


Fig. 2.7: Different types of deflection from different computations. Note the point triplet 1-2-3 is repeated in the figure to improve readability.

2.3. Categorization Parse Tree. After the algorithm computes local point geometry for all interior points, the points are categorized and grouped by categorization. Categorization is carried out in a multi-level process. The levels are labeled on Figure 2.8, which is a recapitulation of Figure 2.2. In Figure 2.8, Level 0 is at the root of the parse tree, the whole trajectory, and deeper levels progressing downward on the parse tree graph, down to Level 3, which has one categorization for each interior point (points 1 through $n-1$).

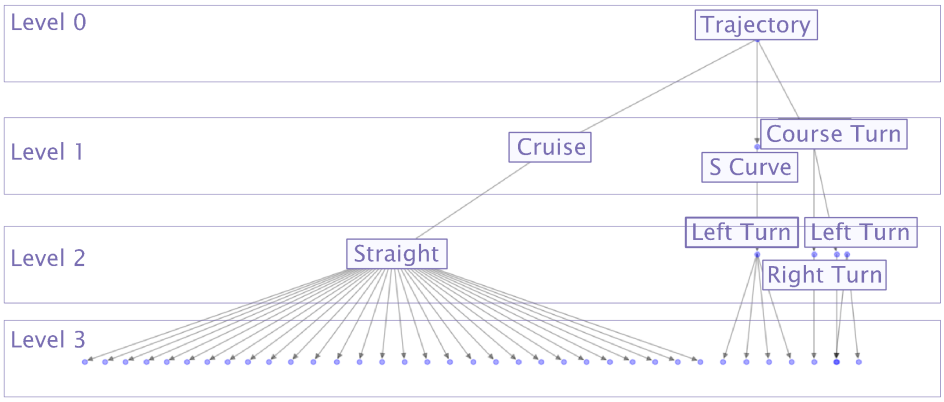


Fig. 2.8: The Parse Tree Graph of Flight A3A458, depicted in Fig.2.1.

2.3.1. Level 3 Processing. The leaves of the parse tree are individual key points. Categorization starts at the bottom of the graph and works its way up. Level 3 processing is carried out first, which is based on the curve data computation of point triplets described above. There is one Level 3 node for each interior point of the trajectory.

Level 3 categorizations are grouped into three aspects: curvature, leg length, and point deflection (δ_{Chord}). Curvature categories are "hard left," "normal left," "flat," "normal right," and "hard right." Deflection categories are "sharp left," "left," "straight," "right," "sharp right." Leg length categories are "short," "medium," "long," "stopped anomaly" (used to indicate double-stamped points), and "jitter anomaly" since radar jitter detection is based on leg length ratios.

2.3.2. Level 2 Processing. Level 2 categorization groups Level 3 nodes according to curvature. Hard left and normal left map to Level 2 category "left." Similarly, hard right and normal right map to Level 2 category "right." Level 3 category flat maps to Level 2 category "straight."

Once processed, any Level 2 node will have between 1 and $n-2$ Level 3 nodes as children. For any Level 2 node, all child nodes must be in the same contiguous sequence. If two Level 3 nodes of the same type are interrupted by a different type, they will be children of different Level 2 nodes.

One may note that mapping Level 3 nodes to Level 2 nodes does not add significant semantic interpretation. The primary purpose of this level of processing is to group like Level 3 nodes into a single Level 2 node. This is necessary because a segment of one kind may include multiple key points which are of the same type.

Level 2 nodes have some geometric properties which are aggregated from the underlying Level 3 geometric information. The most important of these are total point deflection ($\Sigma\delta_{Chord}$) and total leg length. We found that in certain cases partially aliased key points at the boundary between a curved Level 2 segment and a straight Level 2 segment were incorrectly being assigned to the straight segment due to the low value of the transitional curvature. To resolve this issue, a subprocess was added allowing curved Level 2 segments to steal a point from an adjacent straight Level 2 segment. This improved the accuracy of Level 2 curve total point deflection values.

2.3.3. Level 1 Processing. Level 1 categorization groups Level 2 nodes according to multiple patterns in the Level 2 aggregate attributes. Level 1 categories are

1. Cruise: Long, generally straight segments
2. Course Turn: Cruise/Turn/Cruise in which turn $\Sigma\delta_{Chord}$ is less than a U Turn
3. S Curve: A pair of curves in opposite directions with no intermediate straight segment.
4. U Turn: A sequence of straight, turn, straight, in which turn $\Sigma\delta_{Chord}$ is approximately 180°
5. Racetrack: A sequence of turning segments or a single turning segment in which turn $\Sigma\delta_{Chord}$ is greater then 360°
6. Boustrophedon: a sequence containing two or more U Turns in which alternate U turns turn in the opposite direction and straight segments interleave the alternating U Turns
7. No Category: This is the default category and is assigned to Level 1 nodes which do not meet the criteria for other Level 1 categories.

The sequence of Level 1 Categorizations is the primary data product of the CPM. This sequence is viewable in one of the available output types, the parse tree graph. However, the Parse Tree Graph format is not amenable to further analysis. For analysis, the Level 1 Categories are mapped to single characters, which form a string of characters useful for analysis.

2.3.4. Output Files. As described in Section 2, available outputs are

1. a kml file for visualization using Google Earth. (.kml),
2. a plot of the parse tree graph (.png),
3. a detailed arc data report including derived geometry parameters (.csv),
4. a statistical summary report of Level 1 categorizations (.csv),
5. a Level 1 hash bin report grouping all trajectories by a hash string of the Level 1 categorization. (.csv)

An example of item 1, above, is given in Figure 2.1, which is kml output visualized in Google Earth. An example of item 2, the parse tree graph, is given in Figure 2.2. No example of Item 3, a detailed arc data report, is provided.

The first three kinds of outputs are time consuming for the analysis process to generate, so these are not generally created for large batches. More commonly, a large run outputs a batch report, data analysis is performed on that report, and a human selects specific trajectories from the batch report of either item 4 or item 5 to be visualized in one or more of the other three reports for further assessment.

The statistical summary report, item 4 above, consists of 15 columns. One column contains a unique name for each trajectory. Subsequent columns are the number of segments of a given category and the percent of total for each of the seven categories. A subset of the summary output is shown in Table 2.2.

2.3.5. The Hash Bin Output Report. The most useful of the available outputs is the Level 1 hash bin report, listed above as item 5. This report groups trajectories together which have the same Level 1 Categorization sequence.

As described in Section 2.3.3, there are six Level 1 categories plus 'No Category.' We map each of these descriptive names for the categories to a single letter as shown in Table 2.3

For example, if a trajectory is found to have Level 1 categorizations of Course Turn, Cruise, and S-Curve, its hash would be TCS. When a batch run generates a hash bin report, it computes the hash for every trajectory, then it bins all like trajectories into groups based

Name	Cruise Count	Cruise Percent	Turn Count	Turn Percent
07132202_AAL100	4	81.9%	8	16.1%
07171248_AAL1	3	94.5%	5	3.4%
07230452_AAL10	3	91.5%	5	6.7%
09061717_AAL100	3	74.8%	5	20.5%
08250149_AAH5503	3	67.9%	4	16.3%
07092229_AAL1000	3	66.9%	7	25.2%

Table 2.2: Portion of a Statistical Summary Output table for a test run over a large dataset of trajectories.

Level 1 Category	Single Letter Hash
Cruise	C
Course Turn	T
S Curve	S
U Turn	U
Racetrack	R
Boustrophedon	B
No Category	N

Table 2.3: Hash Mapping from Level 1 Category to a single letter.

on their hash. This results is a report in which every trajectory on a given row has the identical Level 1 Categorizations (including order). Table 3.2 shows a portion of a hash bin report.

3. Results. Software development was carried out in Python primarily on a small number of trajectories. Then batch runs were executed on several large datasets.

The primary question we want to ask is, can we use the Curvature Parsing Method to identify trajectories of interest for further investigation by humans from a large real world dataset? It may be noted that what constitutes a "trajectory of interest" is not being defined in this report since the method allows for so many kinds of analysis to be performed.

3.1. Utilizing the Statistical Summary Output Report. To answer the primary question, we ran several queries on the summary data to see what answers might turn up. Upon flagging trajectories of interest, we plotted the results in kml for human inspection.

One batch run resulted in 2,405 flight trajectories. Only the statistical summary output is available for this run. The first query was to look for trajectories with holding patterns by finding trajectories with a high percentage of Racetrack pattern. After the statistical summary output cvs was generated, we sorted the values so that Racetrack appeared at the top of the list.

After reviewing these four flights visually using kml output, we found that one of them, 082214_AAH1, does not contain any racetrack segments. Investigation as to why it incorrectly reports to have 12% racetrack is left as future work.

The other three appear all to be the same flight from Charlotte to JFK, but on different days. In the dataset under study, these three had noteworthy racetrack segments as indicated by the statistical summary output, but the holds do not occur near the destination as is the usual case. Rather they were all near the midpoint of the trajectory over southeastern

Name	Cruise Count	Cruise %	Turn Count	Turn %	No Cat Count	No Cat %	Race-track Count	Race-track %
080118_AAL1	2	68.2%	2	12.9%	0	0.0%	1	18.4%
072818_AAL1	1	34.9%	3	46.1%	2	7.1%	1	13.2%
082214_AAH1	1	77.4%	1	7.5%	0	0.0%	1	12.2%
072517_AAL1	3	42.2%	3	41.9%	1	2.9%	1	9.9%

Table 3.1: Portion of Summary Output Table sorted by Racetrack Percent.

Virginia. A detail of one of these flights, 072818_AAH1, is shown in Figure 3.1.

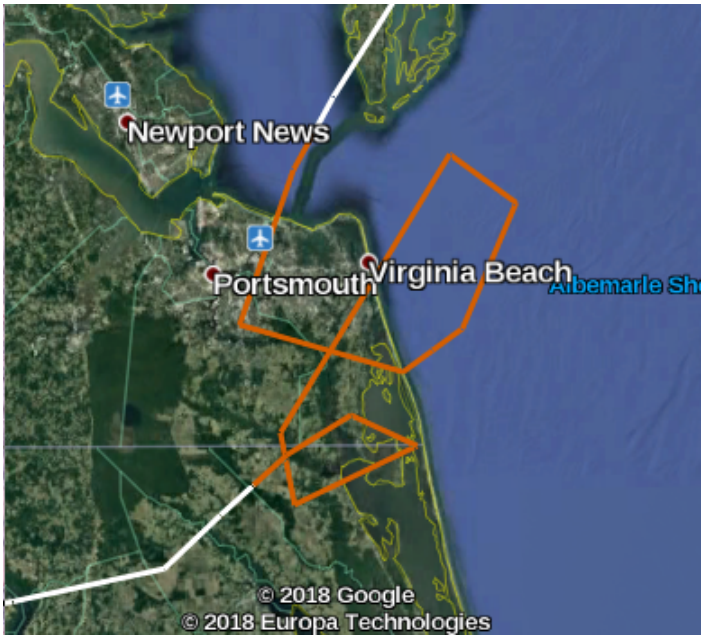


Fig. 3.1: Holding pattern for a flight from Charlotte to JFK. The uninterrupted turning in the same direction is categorized as a Racetrack segment. The trajectory was identified from the statistical summary report sorted by Racetrack Percentage.

3.2. Utilizing the Level 1 Hash Bin Output Report. We carried out a second batch run on a different dataset. In this run we generated a Level 1 Hash Bin report. A total of 21,985 trajectories were processed in 339 seconds resulting in 886 distinct hash combinations. When the report is sorted in descending order of trajectory count per hash, the partial result is shown in Table 3.2. The most common Level 1 categorization hash string is 'C' (Cruise only), with 3,982 trajectories in this category. Not shown in Table 3.2, there are 518 singleton hash strings, which are hash strings with only a single trajectory.

No further data assessment was carried out on the Level 1 Hash Report (L1HR) results. However, one may note that it is in this Hash Report that a type of trajectory similarity emerges. Unlike the statistical summary report, the L1HR preserves the sequence of categories, so the sequence constitutes part of the trajectory similarity. The L1HR aggregates and

Hash String	Trajectory Count	Trajectory IDs
C	3,982	08302100_2XSQM, etc
TC	2,205	09222020_A459, etc
CT	2,114	09041341_9ZHRA, etc
T	1,814	09180011_A5A775, etc
TCT	1,639	09012217_A111971, etc
CTC	1,249	08211613_A1L1958, etc
CTCT	865	08270021_A1, etc
TCTC	660	08030648_AAH11, etc
TCU	462	07141358_AAL100, etc
TCTCT	407	09191922_0UXUB, etc

Table 3.2: Portion of a Hash Bin Output Report sorted by Trajectory Count.

reduces the trajectory data to a form which is better suited for some established data mining approaches.

4. Discussion. The question posed in the Results section has been answered in the affirmative. We are indeed able to use the Curvature Parsing Method to identify trajectories of interest for further investigation. The research into this approach is a partial success. We can get Level 1 Categorizations and perform analysis on these results which help us understand a large dataset, find similarities among temporally and spatially distant trajectories, or quickly find trajectories with specific characteristics.

We claim that this success is only partial because of certain shortcomings. First, we find a number of trajectories which are being classified incorrectly. There may be other bugs in the software which may similarly be considered implementation shortcomings and would be resolved with further work. Another kind of implementation shortcoming is that course correction turns in the middle of long cruises are not being identified as turns.

There are also shortcomings related to the process which must be refined or mitigated. Specifically we refer here to concepts related to curves being missed or incorrectly characterized due to the inherent limitations of sample rate.

Given these observations, we believe a potential for the Curvature Parsing Method to be of value is present. As developed thus far, the Curvature Parsing Method shows promise, but it should be developed more. The Level 1 semantic segments are reasonable and enable an analyst to find alignments with certain features of interest from a large dataset.

4.0.1. Future Research. One indication of the success of the project is that there are multiple directions that future research could take. These directions may be grouped into three themes: Geometry, Categorization, and Analysis.

1. Geometry
   (a) Continue investigation of how to get the most correct arc values for under sampled curves.
   (b) Determine how to find the exact point of curvature (End Tangent/Begin Curve) given edge aliasing tends to obscure these points.
   (c) For racetrack patterns, find the center point which the craft is orbiting. This information could be used to identify Surveillance flights.
   (d) Determine what the best way is to identify radar jitter and the stopped anomaly, and consider improved approaches to ignore or categorize them.
   (e) Add altitude and speed analyses to gain greater understanding of the trajectory.

2. Categorization
   (a) Investigate whether there should be an additional, higher level of categorization. This may be the case given that Boustrophedon depends on a certain sequence of two other Level 1 categories, cruise and u-turn.
   (b) See what other Level 1 categorizations should be added. It would be advantageous to be able to distinguish "loops" from course turns by finding turns between 181° and 360° (similar to a loop ramp at an interchange). It would also be useful to distinguish between "figure eight" and other patterns.
3. Analysis
   (a) Identify useful types of analytic questions which may be asked of the data products.
   (b) What are common patterns in different situations. For example, how often do S-curves appear within the final three category segments, thus indicating a runway alignment maneuver.
   (c) How can machine learning be applied to these processes either to detect previously undetected patterns, or to improve the selections of threshold values at all categorization levels.

5. Conclusion. The Curvature Parsing Method is an innovative approach to identifying and analyzing meaningful subsegments of loosely constrained trajectories. Although it needs more development, we believe that the prospective benefits of this method warrant the additional work.

REFERENCES

[1] S. C. Ahearn and S. Dodge, *Recursive multi-frequency segmentation of movement trajectories (remus*, Methods in Ecology and Evolution, 9 (2018), pp. 1075–1087.
[2] T. Hickerson, *Route location and design*, McGraw-Hill, 1967.
[3] M. D. Rintoul and A. T. Wilson, *Trajectory analysis via a geometric feature space approach*, Statistical Analysis and Data Mining: The ASA Data Science Journal, 8 (2015), pp. 287–301.
[4] M. D. Rintoul, A. T. Wilson, C. G. Valicka, W. P. Kegelmeyer, T. M. Shead, B. D. Newton, and K. R. Czuchlewski, *Panther. trajectory analysis*, (2015).
[5] C. E. Shannon, *Communication in the presence of noise*, Proceedings of the IEEE, 86 (1998), pp. 447–457.
[6] *Aircraft Situation Display To Industry: Functional Description and Interface Control Document*, tech. rep., Volpe Center, 2000.

MACHINE LEARNING FOR LINEAR SOLVERS

CONNOR D. SMITH[†], JOHN KAUSHAGEN[‡], MARK F. HOEMMEN[§], AND CHRIS M. SIEFERT[¶]

Abstract. Physics simulations represent a vital yet often problematic marriage of physics and linear algebra. In order to efficiently run such simulations, analysts require the combination of physics input and input regarding solution techniques for the partial differential equation (PDE). However, analysts are sometimes unable to choose solution parameters that result in a desirable convergence. We present a solution to these inefficiencies through the use of machine learning, in hopes that supervised learning techniques can provide a robust alternative to the often suboptimal linear solver and preconditioner settings chosen by humans.

1. Introduction. Input decks for partial differential equation (PDE) codes involve both "physics" input (which describe the physics of the problem the analyst wishes to solve) as well as parameters governing solution techniques. The further away from the physics these parameters get, the more analysts tend to rely on cut and paste from previous problems. Many times this leads to suboptimal choices or choices which do not let the code run to completion. Moreover, many codes do not allow these non-physics settings to change dynamically, forcing the analyst to pick one set for all timesteps and hope for the best.

In this work, we apply SNL's Avatar framework to a single source of these non-physics parameters, namely the linear solver and preconditioner settings. These parameters are often very far from areas of analyst expertise, yet can have a striking influence on time to complete simulations. Here we will focus on Laplace problems (e.g. electrostatics, or steady-state thermal conduction) solved using the conjugate gradient method (CG) preconditioned with algebraic multigrid (AMG).

Once we generate the required data, we train a classifier using a decision tree ensemble. This classifier is then embedded into the linear solver software, where it is used to recommend optimal parameters to the user. From the application developer's perspective, what is required is a set of features specific to the problem that the user is attempting to solve. Assuming that the classifier has been trained to a sufficient accuracy, the features given by the application will produce a set of solver parameters that will result in a desirable convergence.

2. Machine Learning and Avatar.

2.1. Machine Learning Techniques.

In its most basic sense, machine learning is the process of training machines to use past experience to predict future events. There exist a nearly unlimited number of algorithms that have the ability to accurately predict just about any event that has historical precedence. Some of the more popular machine learning techniques include support vector machines (SVMs) [6], linear and logistic regression [9], neural networks [13], and decision trees [11]. Each of these techniques have strengths and weaknesses, but (as of yet) there is no technique that can universally outperform all of the others. Neural networks in particular represent nearly an entire field by themselves, and continue to accumulate popularity in the machine learning community. For this problem, we experiment with several techniques. We eventually decided on decision trees as the technique of choice, but neural networks also displayed some promise, achieving as high as 90% accuracy. However, due to a stall in accuracy past this point, neural networks were determined as not

[†]Saint John's University, csmith001@csbsju.edu
[‡]Sandia National Laboratories, jkausha@sandia.gov
[§]Sandia National Laboratories, mhoemme@sandia.gov
[¶]Sandia National Laboratories, csiefer@sandia.gov

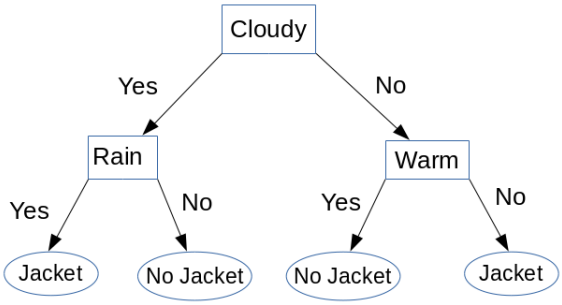


Fig. 2.1: Example of basic decision tree that recommends whether to wear a jacket based on current weather conditions.

suitable. A simple example of a decision tree is Figure 2.1, which demonstrates the process a decision tree would go through to recommend whether or not to wear a jacket.

2.2. Machine Learning Process. While there is no agreed "best" machine learning algorithm, where there does appear to be some consensus within the machine learning community is in the machine learning process, at least when it comes to supervised learning techniques. In order to predict future events, these algorithms require information on past events; in other words, they need data. Thus, the first step in the process is generating data, and if the machine learning algorithm is going to produce accurate predictions, there needs to be a lot of it. For the problem of recommending parameters for linear solvers, we generated 26,040 samples. More often than not, this step in the machine learning process is the most challenging and time consuming.

After generating data, the next step is to train the chosen algorithm on the data. Depending on the algorithm of choice, training can take seconds or days. The actual training process varies from algorithm to algorithm, but usually involves initial guesses on training data followed by refining based on errors made on the training data. Once training is complete, the algorithm is ready to make predictions on data it has yet to see. Next, the third step in the machine learning process is required: testing. Testing involves letting the algorithm predict outcomes of a separate data set, then discerning accuracy based on how well the algorithm performed. If accuracy is determined to be insufficient, one or both of the previous steps are revisited, and in some cases, a completely new plan is put forth.

2.3. Avatar. Decision trees represent a popular type of supervised machine learning algorithm. Given a set of input features, the decision tree algorithm decides on the appropriate label based on previously given labeled data. Depending on the number of features and size of the training set, decision trees can be either very small and simple or large and complex, however decision trees tend to train on and classify data considerably faster than many other machine learning algorithms.

For this problem, we used a set of decision tree tools called Avatar [5]. Aside from its basic functionalities in creating decision trees, Avatar provides the ability to easily create decision tree "ensembles". Instead of one tree to train on and classify data, ensembles are groups of decision trees that, when trained together on the same data, greatly improve classification accuracy. Avatar utilizes other methods of improving accuracy as well, including bagging, boosting, and random subspaces. In bagging [2], the decision tree algorithm builds several different trees on subsets of data sampled with replacement. While each tree is not an accurate classifier by itself, when put together the weak trees tend to produce very accurate

classifiers. Boosting [3] is the process of building an ensemble in which each tree focuses on the samples that the previous trees missclassified. Over time, the boosting ensemble minimizes error on the training set, often greatly improving accuracy as well. The random subspace [8] method is similar to bagging in that it takes random subsets, but instead of sampling the training data, the random subspace method samples the training data features. When applied to the right types of training data, the random subspace method can greatly improve classifier performance.

3. MueLu. MueLu [10] is an algebraic multigrid (AMG) preconditioning package provided in Trilinos [7]. MueLu implements a number of algebraic multigrid algorithms, but for the purposes of this paper we consider smoothed aggregation (SA-AMG) [15]. One of the most difficult parameters for a user to choose when using SA-AMG is the drop tolerance, also known as the aggregation threshold. This is used to remove "weak" connections in a matrix to improve coarse grid quality. It is traditionally defined as follows. A matrix entry, a_{ij}, is dropped if

$$\left| \frac{a_{ij} a_{ji}}{a_{ii} a_{jj}} \right| < \epsilon,$$

for some tolerance ϵ. If this parameter is chosen to be too small, our coarse grids are poor and thus convergence is poor. If this parameter is chosen to be too large, MueLu may crash in the eigenvalue estimation for smoothed aggregation. Thus, analysts using MueLu need to choose a parameter in the "Goldilocks Zone," which is neither too large nor too small. The goal of this work is to use machine learning to guide this decision. For this work, we will be using the distance Laplacian dropping algorithm [4,14], where the drop tolerance is defined on an auxiliary matrix, L, where

$$l_{ij} = \begin{cases} (\mathbf{x}_i - \mathbf{x}_j)^{-2} & \text{if } i \neq j \text{ and } a_{ij} \\ -\sum_{i \neq j} l_{ij} & \text{if } i = j \end{cases},$$

where $\mathbf{x}_i$ represents the mesh coordinates of node i.

4. Problem Features and Correlation. All machine learning algorithms need data to train on, and part of our task was choosing which features to record. There are near limitless choices, and feature choice can be important in the performance of an algorithm. Our problem was heavily dependent on the matrix generated for the problem. Although it would be possible to use information from this matrix as features, we chose to use information about the mesh quality. There has already been much research done on mesh quality and its performance impacts, and seemed a good place to start [1]. We also wanted our features to be dimensionless, scale-invariant, and scalable to a $(0, 1)$ range. For this we settled on element edge length, the material parameter from Laplace ($\nabla \cdot \sigma \nabla \varphi = f$), the determinant of the Jacobian for elements, and stretch, which is defined as $\sqrt{3}$ min edge length/max diagonal length. We recorded other mesh quality metrics as well, but these other metrics proved to be constant for this particular dataset. For each of element edge length and stretch, the minimum and maximum element value were recorded, as well as the mean value. For the Jacobian and material parameter, the ratios of min to mean and max to mean were recorded.

To analyze the independence of our feature choices, we examined the angles between each pair feature vectors. This gave us and idea of how closely related each pair of features was. Figure 4.1 shows the cosine of the angle between each pair of vectors. Along the diagonal are ones since the angle between the same feature vector is zero. There are however some notable relationships. In particular are ElEdge_max and ElEdge_mean which appear to be nearly indistinguishable on the plot. Additionally, there is dependence between Stretch_max

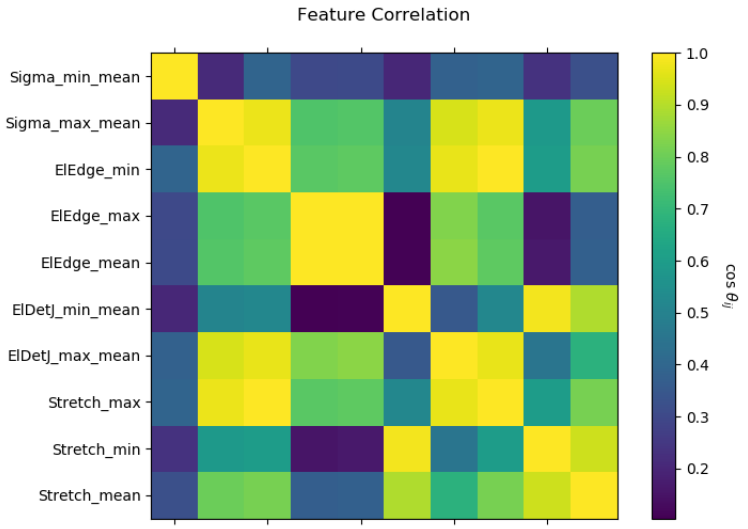


Fig. 4.1: Pairwise cosine of angle between feature vectors. The closer a pairwise relationship is to 1, the more closely related the pair of features.

and ElDetJ_max_mean. Again this is rather unsurprising since stretching should have a noticeable effect on the element Jacobian. Finally one also sees some dependence between Sigma_max_mean and ElEdge_min, although this is most likely an artificial dependence showing up in this dataset.

5. Dataset Description. The dataset we used was created to imitate a normal ALEGRA run [12]. The mesh is a stretched tensor product of a highly refined section and a very coarse section. The material parameter of the mesh is variable in one region of the mesh, and constant everywhere else. Due to the grading, we may obtain very stretched elements on the boundaries of the mesh. In addition, the material parameter varies over different runs.

For each set of stretches and material parameters, eight different drop tolerances were run. The best performance run was considered baseline, and the other seven were compared from the original. In all, we ran 26,040 different combinations.

6. Results.

6.1. Data Classification. Supervised machine learning techniques that classify data into unique categories require training data that has been labeled as one of several possible classes. Data labels can take many forms, ranging from integers to strings of characters. For this problem, data was labeled with an integer value based on a "cost estimate" output from each run in MueLu. Rather than look at run time and deal with the question of performance variability, we use a mathematical estimation of the cost of solving the linear system as our key performance metric. Using an approximation of multigrid work units [16], we first define work as,

$$Work = (SmootherComplexity + 1) \cdot \#Iterations, \tag{6.1}$$

where smoother complexity is the cost of applying smoothers at all levels divided by the cost of applying a matrix-vector product at the fine level. This differs from work units by not including the cost of prolongation/restriction or the cost of vector/scalar operations in the conjugate gradient method. For each problem/parameter combination, we took the work of the *best* parameter on this problem (e.g. the one with the lowest work), and divided this by the work on the problem/parameter pair. This yields a cost estimate, which is 1 for the best algorithm and approaches zero as performance degrades.

Initially, the label on a particular sample was either 0 or 1. A sample was labeled as 0 if its cost estimate was below the arbitrary threshold of 0.8, and labeled as 1 if its cost estimate was above the threshold. A label of 0 represented a "bad" MueLu run, in which the solver either took longer than an ideal time to converge, or crashed before converging. A label of 1 represented a "good" MueLu run, meaning the sample had a set of parameters worth recommending to the user. Later, a third label was introduced: -1. A sample was now labeled as -1 if its cost estimate was negative, which only occurs when convergence is not attained and the program crashes. Therefore, a label of -1 represented the worst case scenario. Aside from adding utility by differentiating between a "bad" run and a "crash", which are significantly different, the addition of a third label surprisingly also saw a slight increase in performance metrics. With both of these benefits, we decided to permanently keep the third "crash" classification.

6.2. Measuring Performance. To measure classifier performance, practitioners usually separate data into training and testing partitions. When the training step is completed, the testing data is used to test the accuracy of the classifier with testing data that has not been seen by the classifier. However, in some cases there is not enough usable data to divide into two separate partitions and still attain good performance in the training step. In these cases, a technique called cross-validation is often utilized. Instead of separating the data into two batches, training is done on the full dataset for N iterations. At each iteration, a subset of the data of size $datasetSize$ / N is set aside, the classifier trains on the rest of the data, and then the classifier is tested on the small subset. Once all iterations are complete, all N of the test results are averaged, resulting in an overall accuracy for the dataset. For our problem, we used both techniques to measure classifier performance.

6.3. Performance Metrics. Accuracy is the most basic measure of classifier performance. It is calculated,

$$\frac{TruePositives + TrueNegatives}{TruePositives + TrueNegatives + FalsePositives + FalseNegatives}, \tag{6.2}$$

where $TruePositives$ are correctly labeled "good" samples, $TrueNegatives$ are correctly labeled "bad" or "crash" samples, $FalsePositives$ are "bad" or "crash" samples incorrectly labeled as "good", and $FalseNegatives$ are "good" samples incorrectly labeled as "bad" or "crash". In other words, accuracy is,

$$\frac{CorrectGuesses}{AllGuesses}. \tag{6.3}$$

There are several metrics aside from accuracy that are vital in understanding classifier performance. In our project, we used both precision and recall as performance metrics. For this particular problem, precision is perhaps the most important metric, as it measures the percentage of samples classified as positive ("good") by the classifier that are correctly labeled as such. This is of high importance to us, because the worst case scenario of the classifier is a "good" classification that, when run in MueLu, actually results in a "crash."

The closer the precision is to 1, the less likely such an incident will occur. Precision is calculated,

$$\frac{TruePositives}{TruePositives + FalsePositives}. \tag{6.4}$$

While not quite as important to this problem, recall is still an interesting metric for our classifier. Recall measures how many of the positive ("good") samples were correctly classified as "good" by the classifier. The closer the recall is to one, the less likely "good" samples will be missclassified as "bad" or "crash." Recall is calculated,

$$\frac{TruePositives}{TruePositives + FalseNegatives}. \tag{6.5}$$

6.4. Classification Performance. To maximize classifier performance, we used all of the previously mentioned decision tree building options (bagging, boosting, and random subspaces) in conjunction. There is not much literature regarding hybrid approaches to decision trees, yet all three of these options together produced the best performance out of our classifier. Between the two data classification approaches, performance metrics were either identical or slightly favored the ternary classification approach. Classification accuracy ended up just above 94 percent which, although less than ideal, should be suitable to accomplish what we need from the classifier. Less suitable to our classifier is the precision, ending up at about 90 percent. While most "good" recommendations will be accurately recommended, the chance that a "good" recommendation is "bad" or "crash" is about 10 percent. However, because of the separation between "bad" and "crash" classifications, we can take a deeper look at what the precision really measures. If we only count "crashes" as negatives, and include "bad" classifications with "good" classifications to make up our positives because "bad" labeled samples still converge, our precision comes out to be about 95 percent. This is important to note, because this means the odds that a worst case scenario will occur, being a "crash" labeled as "good", is only about 5 percent. Table 6.1 summarizes the results of using both three classifications and two.

Number of Classifications	Accuracy	Precision	Recall
Two (0,1)	.9326	.8862	.9002
Three (-1,0,1)	.9431	.9047	.9144

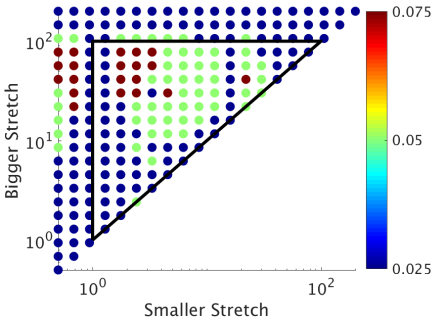
Table 6.1: This table provides the accuracy, precision, and recall for both the ternary and binary classifications.

Figure 6.1 demonstrates an Avatar tool that, instead of increasing accuracy, aids in the visualization of performance results. Called a confusion matrix, this table charts classifier predictions against the true labels of the samples. This allows users to easily identify how well the classifier is performing beyond accuracy, and in this case, to distinguish between "crash" false positives and "bad" false positives. "Crash" false positives are highlighted in red and "bad" false positives are highlighted in orange.

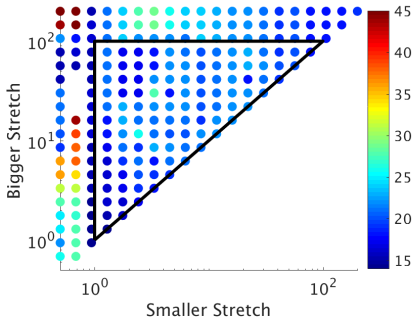
6.5. MueLu Performance. Once embedded in MueLu, the trained classifier predicts the performance of a problem for a set number of drop tolerances, then MueLu recommends a suitable drop tolerance based on the classifier's predictions. Figure 6.2a is a chart of the predicted best drop tolerances for a particular set of problems. Figure 6.2b represents the resulting number of iterations for the problems with drop tolerances recommended by MueLu

		Truth		
		-1	**0**	**1**
Predictions	**-1**	1923	0	348
	0	0	14106	304
	1	394	340	6967

Fig. 6.1: This confusion matrix represents the results of our ternary (three classification) decision tree ensemble. The red number signifies "crash" false positives, which are our worst case scenario. The orange number signifies "bad" false positives, which are not ideal, but still will not crash the linear solver.



(a) Chart of drop tolerances recommended by our classifier. Samples within the black triangle are closely related to our training data, while samples outside of the triangle are not. The axes are labeled in accordance with the stretch sizes used to generate each data point.



(b) Chart of the number of iterations for samples with drop tolerances recommended by our classifier. Spaces that are blank illustrate a crash.

Fig. 6.2: Charts of drop tolerances and number of iterations recommended by our classifier.

in conjunction with our classifier. Samples within the highlighted triangle area come from the same problem domain as our training samples, but as of yet have not been seen by the classifier. As we can see in the chart mapping the number of iterations, samples within the same problem domain performed very well, while samples not as closely related to the training data did not perform as consistently.

7. Conclusions. One of the more complex dilemmas in physics simulations is the choice of certain simulation parameters, in particular the input regarding solution techniques for the partial differential equation. We have provided an alternative to the usual guess work associated with choosing these parameters through the use of machine learning. While we have so far only produced results for the multigrid drop tolerance parameter, our method for recommending parameters is applicable to a nearly endless array of solution parameters. We have successfully demonstrated that the combination of human and machine decision making has the ability to greatly improve efficiency in physics simulations, and is a promising field of study moving forward.

REFERENCES

[1] *Metrics for hexahedral elements.* `https://cubit.sandia.gov/public/13.2/help_manual/WebHelp/mesh_generation/mesh_quality_assessment/hexahedral_metrics.htm`.
[2] L. Breiman, *Bagging predictors*, Machine Learning, 24 (1996), pp. 123–140.
[3] Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, Computer and System Sciences, 55 (1997), pp. 119–139.
[4] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala, *ML 5.0 smoothed aggregation user's guide*, Tech. Rep. SAND2006-2649, Sandia National Laboratories, 2006.
[5] L. O. Hall, K. W. Bowyer, N. V. Chawla, T. E. Moore, and W. P. Kegelmeyer, *Avatar – adaptive visualization aid for touring and recovery*, Tech. Rep. SAND2000-8203, Sandia National Laboratories, 2000.
[6] M. Hearst, *Support vector machines*, IEEE Intelligent Systems and their Applications, 13 (1998), pp. 18–28.
[7] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, *An Overview of Trilinos*, Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.
[8] T. K. Ho, *The random subspace method for constructing decision forests*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (1998), pp. 832–844.
[9] J. Neter, M. H. Kutner, W. Wasserman, and C. J. Nachtsheim, *Applied Linear Regression Models*, McGraw-Hill Irwin, 2004.
[10] A. Prokopenko, J. J. Hu, T. A. Wiesner, C. M. Siefert, and R. S. Tuminaro, *MueLu users guide 1.0*, Tech. Rep. SAND2014-18874, Sandia National Labs, 2014.
[11] J. Quinlan, *Simplifying decision trees*, International Journal of Man-Machine Studies, 27 (1987), pp. 221–234.
[12] A. C. Robinson and W. J. Rider, *Alegra: An arbitrary lagrangian-eulerian multimaterial, mutiphysics code*, in Proceedings of the 46th AIAA Aerospace Sciences Meeting, January 2008.
[13] D. F. Specht, *Probabilistic neural networks*, Neural Networks, 3 (1990), pp. 109–118.
[14] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, San Diego, 2001.
[15] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.
[16] P. Wesseling, *Introduction to Multigrid Methods*, John Wiley and Sons Ltd., 1992.

Applications

Articles in this section discuss the application of computational techniques to simulate physical systems.

Hedge, Van Bloemen Waanders, Littlewood, and Brown-Shaklee support the production of defect-free parts in additive manufacturing by developing predictive models for ceramic components. Using peridynamics, they model thermal debinding and sintering processes. They demonstrate that their model captures several key features of defects in the experimental specimen.

Meredith, McGregor, and Kramer address the utility of simulation-based parameter estimation where analytic studies are impossible and experimental investigations are impractical. They describe a method for determining the convolution kernel of an antenna using electromagnetic simulation. Their method permits the prediction of the antenna response to any arbitrary time-dependent input signal.

Brickson, Jacobson, and Baczewski implement the effects of magnetic fields and spin-orbit coupling in an interior penalty discontinuous Galerkin discretization of Schrödinger-like equations. Their method enables simulations of qubit device design.

Sagredo and Cangi develop an exchange-correlation functional for density functional theory which captures bulk, surface, and confinement physics for materials modeling of d- and f-electron systems. They demonstrate its accuracy on the jellium surface.

A. Cangi
M. L. Parks

December 6, 2018

A PERIDYNAMIC MODEL FOR DIRECT WRITE

ARUN S. HEGDE*, BART G. VAN BLOEMEN WAANDERS†, DAVID J. LITTLEWOOD‡, ADAM W. COOK§, AND HARLAN J. BROWN-SHAKLEE¶

Abstract. Developing predictive models for the additive manufacturing of ceramic components is essential to producing defect-free parts in a reliable manner. In this report, we investigate peridynamics, an alternative theory to classical continuum mechanics, as a foundation for developing such models. Our focus is on direct write, a subfamily of additive manufacturing processes in which parts are produced by layering a ceramic slurry to create the green state and then applying heat in two steps, first to dry the green state (thermal debinding) and second to solidify the resulting powder compact (sintering). Simple models for the first two stages – green state and thermal debinding – are constructed using the peridynamic theory. The efficacy of the strategy is demonstrated through comparison with experiment, with the models capturing several key features of defects in the experimental specimen.

1. Introduction. Developing realistic and predictive models for additive manufacturing requires multiscale and multiphysics modeling techniques. The production of ceramic parts is an important example of this. In a typical scheme, a ceramic powder is mixed with a polymeric binding agent to create a thixotropic and pliable slurry, which is then layered onto a substrate to achieve a prescribed geometry. The polymer binder is removed from the resulting green state via an initial heating, leaving only the shaped ceramic powder. After the binder is removed, the shaped powder is sintered into its final state. The end goal of this process is to produce a part that exhibits the desired material properties without defects or anomalies. Unfortunately, the reality is that material properties of the manufactured parts can be less than reliable and often display considerable variation [15]. Moreover, significant amounts of defects, such as cracks or voids, can be detected when viewing a finished product at different length scales. Each of these subprocesses – the green state, removal of the binder, and sintering – could contribute to the formation of defects. In order to mitigate these challenges, computational models must be able to reliably predict when and how defects occur, and therefore must account for the various physics and length scales involved in the process.

In the present work, we focus our attention on direct write (DW), a subfamily of additive manufacturing processes in which three-dimensional parts are constructed in a layered fashion by extruding material through a deposition tool, such as a syringe or nozzle [8,10,11]. DW consists of a sequence of four stages: (1.) creating a homogeneous slurry with the correct rheological behavior, (2.) using an optimal amount of pressure and heat at the syringe for deposition onto the substrate, (3.) applying an optimal heating schedule to extract the binding agent, and (4.) applying the right amount of heat for sintering. A proper setting of multiple decision variables and parameters during each of these stages is required to produce ceramic parts with the desired geometric features and predictable material properties. Although this motivates an optimal control problem, our first challenge is to develop a forward model that not only captures the general characteristics of the process, but also spans the different, critical subprocesses. The approach we follow is rooted in the peridynamic theory, which provides a flexible computational mechanics framework in which the formation of defects is a natural consequence of the governing equations. Although a range of mechanical, thermal, and electrical properties are potential design targets, we

*University of California, Berkeley – Department of Mechanical Engineering, arun.hegde@berkeley.edu
†Sandia National Laboratories, bartv@sandia.gov
‡Sandia National Laboratories, djlittl@sandia.gov
§Sandia National Laboratories, acook@sandia.gov
¶Sandia National Laboratories, hjbrown@sandia.gov

concentrate on the characterization of defects throughout the DW process. Specifically, we seek to address the following question: can the peridynamic framework help us construct a model that is (a.) capable of spanning the green state, binder removal, and sintering steps of DW and (b.) flexible enough to generate defects with features similar to those observed in experimental data? Our results suggest positive answers to both questions.

Peridynamics was first introduced by Silling in [17] as an alternative theory for solid mechanics that naturally handled length scale effects and subsumed the modeling of discontinuities, which traditionally required a separate set of theory and computational techniques. This was accomplished by replacing spatial derivatives, a key feature of the classical theory, with integral equations defined over neighborhoods. In the current work, we utilize the meshfree discretization of Silling and Askari [18], as implemented in the *Peridigm* simulation code [14]. Peridynamics has successfully been applied in a number of settings, including prediction of crack growth [1,9], realistic fracture modeling [2,25], gas and explosive modeling [5], heat and mass transfer [3,13], fluid-structure interactions [23], and shockwave and impact modeling [21]. Recent work by Silling and coworkers [16] extended peridynamics to formulate a novel multiscale approach to sintering. These developments motivate the present work and suggest that the modeling of the various stages of DW can be unified under a single peridynamic framework.

The remainder of the paper is organized as follows. Section 2 provides a brief overview of peridynamics and discusses specific material and damage models. In Section 3, we use the theory to formulate a forward model for DW focusing on the green state and removal of the binder. Throughout the section, we emphasize the flexibility and utility of the approach by highlighting design decisions that simplify the modeling procedures. The developed forward model spans the green state and thermal debinding stages of DW and is compared to experimental data from sintered specimen in Section 4. In particular, we emphasize the similarities between simulated and experimentally-observed defects.

2. Peridynamics. In this section, we provide a brief introduction to peridynamics. For more detailed accounts of the theory, please refer to work by Silling and coworkers [17,19,20].

2.1. General formulation. Peridynamics was initially developed as a nonlocal reformulation of classical continuum mechanics [17]. Whereas the classical theory constructs models based on interactions between material points and their infinitesimal neighbors, i.e., via spatial derivatives, peridynamics is based on interactions between material points and their neighborhoods. Let $\mathcal{B} \subseteq \mathbb{R}^d$ (typically $d = 3$) denote a reference body and let $\mathbf{x} \in \mathcal{B}$. For notational convenience, boldfaced variables are used to denote vectors in $\mathbb{R}^d$. Given $\delta > 0$, define the neighborhood of the material point $\mathbf{x}$ to be $\mathcal{H}_{\mathbf{x}} = \{\mathbf{x}' \in \mathcal{B} : \|\mathbf{x}' - \mathbf{x}\|_2 < \delta\}$, i.e., the intersection of an open ball centered at $\mathbf{x}$ with radius δ and the reference body $\mathcal{B}$. The parameter δ is often referred to as the *horizon* as it dictates with which points $\mathbf{x}$ may interact. If $\mathbf{x}' \in \mathcal{H}_{\mathbf{x}}$, then $\mathbf{x}$ and $\mathbf{x}'$ are said to share a *bond*, which is denoted by the corresponding relative position vector $\mathbf{x}' - \mathbf{x}$.

It is important to note that both neighborhoods and bonds are determined with respect to proximity in the reference configuration. For any $t \geq 0$, let $\mathbf{u}(\mathbf{x}, t)$ denote the displacement vector between the point $\mathbf{x}$ in the reference configuration and its new location at time t. Whether two deformed points $\mathbf{x} + \mathbf{u}(\mathbf{x}, t)$ and $\mathbf{x}' + \mathbf{u}(\mathbf{x}', t)$ are bonded depends on whether the bond $\mathbf{x}' - \mathbf{x}$ exists in the reference configuration. With this setup, discontinuities such as cracks or voids are directly modeled by deleting bonds when a failure criterion is met, for example, if the deformed bonds have stretched too far.

The peridynamic equation of motion can be derived as,

$$\rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x},t) = \int_{\mathcal{H}_\mathbf{x}} \left(\underline{\mathbf{T}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle - \underline{\mathbf{T}}[\mathbf{x}',t]\langle\mathbf{x}-\mathbf{x}'\rangle\right) dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x},t) \tag{2.1}$$

for all $\mathbf{x} \in \mathcal{B}$, where $V_{\mathbf{x}'}$ refers to the standard volumetric measure. The quantity $\rho(\mathbf{x})$ is the density field of the reference body and $\mathbf{b}(\mathbf{x})$ is a prescribed volumetric force applied at the material point $\mathbf{x}$. $\underline{\mathbf{T}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle$ describes the *force state* associated with the point $\mathbf{x} \in \mathcal{B}$ at time t. A force state is an important example of a *peridynamic state*, which is a scalar, vector, or tensor valued operator that takes bonds as input. Peridynamic states are typically denoted by the underline convention with the bond being operated on indicated between angle brackets $\langle\cdot\rangle$. Any spatial and/or temporal dependence is specified using regular brackets $[\cdot]$. Hence, $\underline{\mathbf{T}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle$ is a nonlinear function that depends on a reference position and time, takes as input a bond (in the neighborhood of the reference position), and returns a d-dimensional vector of force densities with units of force per unit volume squared. A more comprehensive discussion of state-based peridynamics can be found in [19].

Another important example of a peridynamic state is the *deformation state*,

$$\underline{\mathbf{Y}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle = \mathbf{y}(\mathbf{x}',t) - \mathbf{y}(\mathbf{x},t) \tag{2.2}$$

where $\mathbf{y}(\mathbf{x},t) = \mathbf{x} + \mathbf{u}(\mathbf{x},t)$ represents the deformed position of point $\mathbf{x}$ at time t. Essentially, the deformation state operates on a bond $\mathbf{x}'-\mathbf{x}$ and returns the corresponding deformed bond at time t. The associated *extension scalar state* is a scalar-valued expression defined as,

$$\underline{\mathbf{e}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle = \|\underline{\mathbf{Y}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle\| - \underline{x}[\mathbf{x}]\langle\mathbf{x}'-\mathbf{x}\rangle \tag{2.3}$$

where $\underline{x}[\mathbf{x}]\langle\mathbf{x}'-\mathbf{x}\rangle = \|\mathbf{x}'-\mathbf{x}\|_2$ is the *reference scalar state*. Hence, (2.3) measures the change in bond length after deformation. Normalizing this quantity by $\underline{x}[\mathbf{x}]\langle\mathbf{x}'-\mathbf{x}\rangle$ provides the dimensionless strain in the indicated bond. As will be illustrated shortly, different material models are specified by different expressions for the force state, which in turn depend on the deformation state, i.e., $\underline{\mathbf{T}}$ can be written as a function of $\underline{\mathbf{Y}}$.

Numerical solution of (2.1) requires both spatial and temporal integration. In the present work, we use the software *Peridigm* [14], an open-source computational peridynamics code which computes (2.1) using the meshfree method of Silling and Askari [18]. Spatial integration is carried out by partitioning the reference body $\mathcal{B}$ into m disjoint regions, represented by the collection of nodes and volumes $\{(x_i, \Delta V_i)\}_{i=1}^m$ where x_i is some representative point from region i and ΔV_i is the volume of region i. For each region, let the collection of δ-neighboring elements be defined as,

$$\mathcal{F}_i = \{p : \|\mathbf{x}_p - \mathbf{x}_i\|_2 \leq \delta\}, \quad i = 1, ..., m. \tag{2.4}$$

The spatially discretized equation of motion can therefore be written as,

$$\rho(\mathbf{x}_i)\ddot{\mathbf{u}}(\mathbf{x}_i,t) = \sum_{p\in\mathcal{F}_i} \left(\underline{\mathbf{T}}[\mathbf{x}_i,t]\langle\mathbf{x}_p-\mathbf{x}_i\rangle - \underline{\mathbf{T}}[\mathbf{x}_p,t]\langle\mathbf{x}_i-\mathbf{x}_p\rangle\right)\Delta V_p + \mathbf{b}(\mathbf{x}_i,t), \tag{2.5}$$

for each $i = 1, ..., m$. Under the assumption that the force state is Riemann integrable, the discretization becomes quite natural. Following [18], time integration can be carried out using a central difference scheme, which leads to a fully discretized equation of motion,

$$\begin{aligned}\rho(\mathbf{x}_i)\frac{\mathbf{u}(\mathbf{x}_i,t_{k+1}) - 2\mathbf{u}(\mathbf{x}_i,t_k) + \mathbf{u}(\mathbf{x}_i,t_{k-1})}{(\Delta t)^2} = \\ \sum_{p\in\mathcal{F}_i} \left(\underline{\mathbf{T}}[\mathbf{x}_i,t_k]\langle\mathbf{x}_p-\mathbf{x}_i\rangle - \underline{\mathbf{T}}[\mathbf{x}_p,t_k]\langle\mathbf{x}_i-\mathbf{x}_p\rangle\right)\Delta V_p + \mathbf{b}(\mathbf{x}_i,t_k), \qquad \forall i = 1, ..., m\end{aligned} \tag{2.6}$$

where $\{t_k\}_{k=1}^n$ is a discretization of the interval $[0, T]$ for some $T > 0$ with stepsize Δt. The above equations can be solved when appropriate initial and boundary conditions are provided.

2.2. The linear peridynamic solid model. In this subsection, we describe the linear peridynamic solid (LPS) material model derived in [19]. Essential, the LPS model provides a nonlocal interpretation of classical linear elastic theory. The LPS principle states that the force density vector produced by $\underline{\mathbf{T}}[\mathbf{x}, t]\langle \mathbf{x}' - \mathbf{x}\rangle$ acts along the direction of deformation state. Specifically,

$$\underline{\mathbf{T}} = \underline{t}\frac{\underline{\mathbf{Y}}}{\|\underline{\mathbf{Y}}\|}, \tag{2.7}$$

where $\underline{t}$ is a scalar valued peridynamic state. Note, the input arguments to the peridynamic states, $[\mathbf{x}, t]\langle \mathbf{x}' - \mathbf{x}\rangle$, have been suppressed for readability. In a general setting, a material that obeys (2.7) is termed *ordinary*. The magnitude $\underline{t}$ is defined as,

$$\underline{t} = \frac{-3p}{m}\underline{\omega}\,\underline{x} + \frac{15G}{m}\underline{\omega}\,\underline{e}^d \tag{2.8}$$

where p is the peridynamic pressure and G is the shear modulus. The peridynamic pressure is generally given by $p = -K\theta$, where K is the bulk modulus and θ is the dilatation. The remaining components of (2.8) are defined as follows.

The weighted volume m is a spatially varying quantity that only depends on the reference configuration and is given by,

$$m[\mathbf{x}] = \int_{\mathcal{H}_\mathbf{x}} \underline{\omega}\,(\underline{x})^2\, dV_{\mathbf{x}'}. \tag{2.9}$$

The *influence function* $\underline{\omega}$ is a nonnegative function that weights bonds in the neighborhood $\mathcal{H}_\mathbf{x}$, thus determining their contribution to the integral. If $\underline{\omega}$ is solely a function of the scalar $\underline{x}$, then the influence function is spherical and the material is isotropic. The dilatation is defined as,

$$\theta[\mathbf{x}, t] = \frac{3}{m}\int_{\mathcal{H}_\mathbf{x}} \underline{\omega}\,\underline{x}\,\underline{e}\, dV_{\mathbf{x}'}. \tag{2.10}$$

The final term in (2.8), $\underline{e}^d$, is the *deviatoric part* of the extension state $\underline{e}$ and is given by $\underline{e}^d = \underline{e} - \theta\underline{x}/3$. The effect of changes in temperature-dependent problems can be easily incorporated by subtracting the thermal expansion from the extension scalar state [12],

$$\underline{e}^\star = \underline{e} - \alpha\Delta T\underline{x} \tag{2.11}$$

where α is the material's linear coefficient of thermal expansion and ΔT is the temperature change. The new extension state $\underline{e}^\star$ then replaces $\underline{e}$ in the preceding formulas.

2.3. The critical stretch damage model. An important feature of the peridynamic framework is that discontinuities such as cracks or fractures are handled naturally by the theory. Essentially, these forms of damage occur when two points $\mathbf{x} \in \mathcal{B}$ and $\mathbf{x}' \in \mathcal{H}_\mathbf{x}$ cease to interact, i.e., the bond $\langle \mathbf{x}' - \mathbf{x}\rangle$ is removed and the load that it would have carried is distributed among neighboring bonds. These neighboring bonds are more likely to fail as they must now support the extra load. In this fashion, damage can emerge realistically and propagate autonomously through a material. The fidelity of this scheme has been observed in

a number of studies, e.g., [1,5,9]. The critical stretch model for bond failure is one approach to modeling the formation and propagation of discontinuities [18]. Let s_0 denote the *critical stretch* for bond failure and define the *bond stretch* to be the peridynamic state,

$$\underline{s} = \frac{\underline{e}}{\underline{x}} \tag{2.12}$$

which is simply the normalized extension. Define $\underline{\mu}$ as follows,

$$\underline{\mu} = \begin{cases} 1 & \text{if } \underline{s} < s_0 \text{ for all } 0 \le t' \le t \\ 0 & \text{otherwise} \end{cases}. \tag{2.13}$$

The motivation for this definition is that bond failure is irreversible; $\underline{\mu}$ takes on the value of 1 only if the bond stretch never equals or exceeds the critical stretch, i.e., the bond remains undamaged for all time $t' \le t$. From this, the notion of damage at a particular point can be introduced by integrating over all bonds in the neighborhood of that point,

$$\phi(\mathbf{x}, t) = 1 - \frac{\int_{\mathcal{H}_\mathbf{x}} \underline{\mu}[\mathbf{x}, t]\langle \mathbf{x}' - \mathbf{x}\rangle dV_{\mathbf{x}'}}{\int_{\mathcal{H}_\mathbf{x}} dV_{\mathbf{x}'}}. \tag{2.14}$$

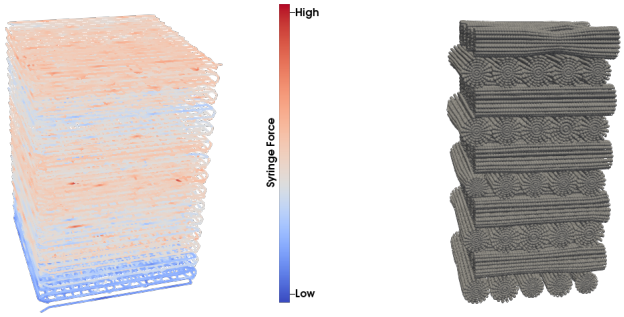
Hence, $\phi(\mathbf{x}, t)$ measures the fraction of material points in the neighborhood of $\mathbf{x}$ for which no bonds exist (at time t). If $\phi(\mathbf{x}, t) = 1$, then $\mathbf{x}$ is totally disconnected from its neighbors. Damage is incorporated in both (2.1) and (2.6) by introducing $\underline{\mu}$ as a multiplier to the force state $\underline{\mathbf{T}}$.

3. A forward model for direct write. In this section, peridynamic theory is used to construct a forward model spanning the green state, debinding, and sintering stages for the direct write (DW) assembly of ceramic parts. We begin with a description of the DW fabrication technique. In a typical application, a ceramic powder is mixed with a sacrificial polymeric binder in order to create a workable slurry that can be shaped into complex patterns or arrangements at the right pressures and temperatures. This shaping occurs via a controlled procedure where the slurry is deposited onto the substrate by a deposition tool, such as a syringe or nozzle, in a layered fashion to achieve the target geometrical configuration. The manner in which the deposition occurs can be broken into two principal categories [11]: filamentary-based approaches and droplet-based approaches. Our focus is on filamentary-based approaches where the slurry is extruded through the nozzle in continuous strands.

Once deposited, the material is referred to as being in the *green state* and the shaping of the material is complete. The remaining stages can be thought of as post-processing steps in order to give the final part desirable mechanical properties. The binding agent is first removed from the green state in a process known as *debinding*. In the present work, we focus on thermal debinding, which is probably the most common industrial technique for polymer removal [6]. During this procedure, the polymeric binding agent is removed from the green state through controlled heating, leaving only the ceramic in the form of a powder compact, i.e., a porous packing of loose grains. The resulting part is then sintered in order to further fuse the powder grains together and create a strengthened and dense solid part. Note that the removal of the binder before sintering is a critical step – if debinding is not carried to completion, the residual polymer left in the part may act as a contaminant during the subsequent sintering and lead to a final part with weakened material properties [22]. A simple example to have in mind is clay pottery – clay and water are mixed together into a

malleable form, shaped, and then heated in a kiln to produce a pot. At any stage of the above processes, flaws in the part could develop and grow into defects in the final product. The modeling of these defects is the critical aspect of the current work.

3.1. Green state. The term green state refers to the configuration of the material before heating, i.e., the state after the material is deposited but prior to debinding and sintering. In order to simplify the construction, we do not explicitly model the mixing of the slurry and the deposition onto the substrate. Rather, the green state is constructed by arranging the slurry strands according to specified pressure maps, which contain the trajectory ($\mathbf{x}(t) \in \mathbb{R}^3$ coordinates), temperatures and syringe forces associated with the printing procedure. As such, the model for the green state becomes a direct implementation of the theory described in Section 2.2 and Section 2.3 for the geometry specified by the pressure maps. An example pressure map displaying the nozzle trajectory and syringe forces is shown in Figure 3.1a .



(a) (b)

Fig. 3.1: Left, example pressure map detailing the laydown path of the slurry and syringe forces at the nozzle. Right, visualization of the green model containing multiple strands from a subset of the pressure map.

In order to assemble the green state, we have to make some critical modeling assumptions. Consider an individual slurry strand extending linearly from the coordinate $\mathbf{x}_L$ to $\mathbf{x}_U$ and let $[\mathbf{x}_L, \mathbf{x}_U]$ denote the corresponding line segment. Let $f(\mathbf{x})$ be the syringe force associated with the pressure map. The strand is modeled as a cylinder with radius depending on the syringe force,

$$r(\mathbf{x}) = r_0 + \beta \left(\frac{\mathbf{x} - \mathbf{x}_L}{\mathbf{x}_U - \mathbf{x}_L} f(\mathbf{x}_U) + \frac{\mathbf{x}_U - \mathbf{x}}{\mathbf{x}_U - \mathbf{x}_L} f(\mathbf{x}_L) - f_{\text{ave}} \right) \qquad \text{for } \mathbf{x} \in [\mathbf{x}_L, \mathbf{x}_U] \tag{3.1}$$

where r_0 is the nominal radius, β is a sensitivity parameter determining the variations among strands based on force measurements, and f_{ave} is the average syringe force (over the entire pressure map). Essentially, the strand radius at a position $\mathbf{x}$ is given by a nominal radius plus the scaled deviation between a linear interpolation of the forces at the strand endpoints and the average force, reflecting the fact that larger syringe forces correspond to greater quantities of material being deposited. The strand endpoints and forces are typically extracted from sequential data points in experimentally-recorded pressure maps. Additionally, the nominal strand radius r_0 and the constant β are unknown model

parameters that must be estimated from experimental data. It should be emphasized that (3.1) was chosen for its simplicity; model forms of greater complexity may lead to more realistic results. As will be demonstrated in Section 4, this model is enough to capture several prominent features of experimental samples. An example strand model corresponding to a subset of the pressure map in Figure 3.1a with $\beta = 0.08$ is presented in Figure 3.1b.

A full numerical implementation of the green state can be constructed by combining the pressure map and strand model, discretizing the resulting geometry into nodes as in Section 2.1, choosing a horizon δ, implementing the force states associated with the linear peridynamic solid model of Section 2.2, and solving the resulting set of equations. This task is made simple in Peridigm [14].

3.2. Thermal debinding. After the green state has been constructed, the green part enters a thermal debinding stage in which the polymer binder is removed from the material via a prolonged period of heating. This process, sometimes referred to as *burnout* or *burnoff*, involves an intricate combination of chemical and physical mechanisms that can often result in anomalies such as bloating, blistering, cracking, and void growth [6, 22]. As temperature increases, the polymer degrades and eventually decomposes into volatile species that diffuse to the surface of the part. When these gasses become trapped, the resulting internal stresses can lead to defect formation [22]. Enneti and coworkers [6] posit that the increased internal pressure during thermal debinding is exacerbated by the angular shape of ceramic particles, which restricts how freely the particles may rearrange and thus acts as a further impediment to the escaping polymer. An additional complication is the distribution of internal pressure throughout the part could be uneven as the composition of the slurry strands may be highly variable. For example, polymer-rich regions of the green state can form into pores and voids.

The above discussion suggests that the principle mechanism for defect formation during thermal debinding is an increased internal pressure brought about by an increased temperature. Rather than explicitly modeling the complex array of mechanisms that contribute to the internal pressure, e.g. evaporation, fluid flow through porous media, pyrolysis, etc., we choose to construct a simple peridynamic material model that exerts a user-specified time-dependent pressure on its surroundings. This material can be placed among the arranged slurry strands to emulate an increasing internal pressure as the green state undergoes a temperature rise. In our implementation, we utilize the peridynamic material model developed by Turner [23] for representing a porous material with fluid pressure accumulated in the pores. The new material model requires a simple modification to (2.8), with the peridynamic pressure now containing an additional contribution from the fluid pressure p_f, which may vary in time or space, i.e.,

$$p = -K\theta + \gamma p_f \tag{3.2}$$

where γ is the fluid pressure coefficient. We refer to the Peridigm user guide [14] for discussion on the software implementation of new material models. An obvious challenge with this approach is that the internal pressure is usually unmeasured and therefore unknown. To handle this, we can parameterize the pressure function and treat the parameters as quantities to be estimated or calibrated.

3.3. Sintering. Once the polymer is removed from the green state, the part is sintered into its final state. Sintering is a thermal process in which a powder compact made up of individual powder grains is converted into a solid mass through the application of heat. This process is fundamentally driven by a reduction in surface energy of the powder grains [4, 7]. We first note that the surface energy is created by the disrupted

atomic bonds on free surfaces of the grains and is hence related to the area density of broken bonds. At grain boundaries, i.e., the interfaces where two grains are in contact, the broken bonds of both contact surfaces partly align and connect, leading to high grain boundary energy if the alignment is poor (more broken bonds) and lower grain boundary energy otherwise (better alignment, fewer broken bonds). As the temperature is increased, atoms in the grains become more mobile and curvature gradients due to the mixed particle-pore structure promote mass flow within contacting grains. This mass flow tends to fill and smooth out the connecting regions between adjacent grains, a phenomenon known as neck formation. In this manner, grain boundaries grow during sintering, leading to a decrease in free surface areas and hence a reduction of free surface energies. The result is a densified, solid part where the formerly loose grains have fused together.

Due to unavoidable variations in grain sizes and shapes, as well as anomalies either present in the green state or formed during debinding, the ceramic powder compacts are rarely uniform and may contain voids and other defects which grow into larger-scale flaws during sintering. For example, preexisting voids may merge rather than close, leading to the final part containing even larger voids. Moreover, it has been observed that sintering tends to amplify defects developed during thermal debinding [6]. The key motivation for a peridynamic model for sintering comes from the importance of peridynamic bond forces. Recall from Section 2, peridynamic theory is built on nonlocal force interactions between material points within the same horizon. As a result, the framework provides a natural setting to model the long-range forces responsible for powder grains attracting and deforming. In work by Silling and coworkers [16], a multiscale model for sintering is proposed that characterizes the aforementioned surface energies through intergrain and intragrain peridynamic bonds rather than atomic bonds. Importantly, this approach has been demonstrated to reproduce several microstructural features that appear in real-world sintering applications.

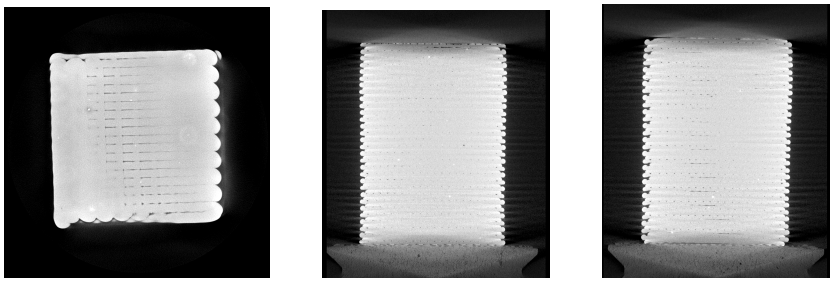
The full forward model is constructed by connecting the models for the green state, debinding, and sintering in series. Hence, the forward model is initialized at the green state, which then undergoes thermal debinding, and the output of thermal debinding is passed as an input to sintering. Importantly, each stage of the model is underpinned by the same fundamental peridynamic theory.

4. Results. The forward model developed in Section 3 utilizes the peridynamic framework to span the green state, thermal debinding, and sintering stages of DW. This answers the first half of the question posed in the introduction. Now, we pursue the second half, namely, how well does the forward model capture defects observed in real-world systems? In what follows, we highlight the flexibility of the peridynamic approach by discussing design decision that allow us to reproduce interesting features of the experimental specimen. It is important to note that the results presented in this section constitute only a partial comparison between simulation and experiment. The simulation results are taken after running the green state and thermal debinding components of the code and comparisons are made with selected defects from sintered specimen which were judged to have originated during debinding. Hence, the conclusions drawn in this section are primarily qualitative and motivate more quantitative comparisons in the future.

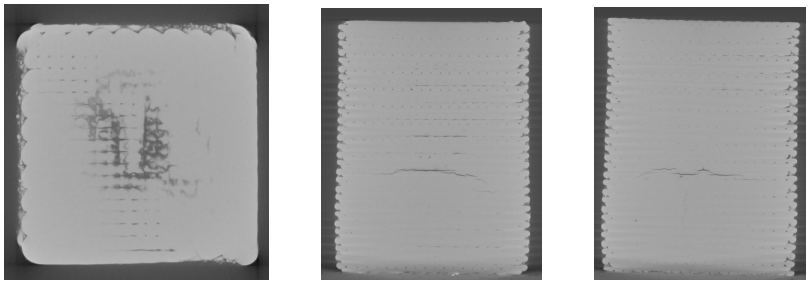
In the examples presented below, the slurry mixture was composed of 60% alumina powder and 40% polymer, with trace amounts of stearic acid. The experiment followed the procedures highlighted in Section 3 – the slurry was printed onto a substrate, the

green state underwent thermal debinding, and the resulting powder compact was sintered. Experimental data was available in two forms. First, numerical measurements were taken during the printing procedure. The corresponding pressure map was shown previously in Figure 3.1a and served as an input to the green state peridynamic model. Second, three-dimensional computed tomography (CT) scans of entire experimental part were taken at both the green state and post-sintering. Unfortunately, CT scans during and after thermal debinding were unable to be collected due to the fragility of the powder compact. On the simulation side, the horizon was selected to be roughly thrice the minimum distance between nodes, as recommended in [14]. The pressure function to emulate the internal pressure during thermal debinding was represented as a sigmoid function, with the peak pressure an unknown parameter. In the simulation results displayed below, the unknown parameters – including peak pressure, the strand model parameter β, and the horizon δ – were essentially fixed at convenient values and tuned by hand. This is justified by the fact that our goal here is not in parameter estimation, but rather in assessing the flexibility of the peridynamic framework in capturing qualitative features of defects. The results of this study motivate more quantitative comparisons between model and data, e.g., by using calibration or inversion techniques. This avenue of investigation is further motivated in the study conducted by Turner and coworkers [24], which presents an adjoint-based method for handling inverse problems with peridynamic models.

Cross sections of the experimental part at both the green state and after sintering are displayed below in Figure 4.1.



(a) Green XY slice (b) Green YZ slice (c) Green XZ slice



(d) Sintered XY slice (e) Sintered YZ slice (f) Sintered XZ slice

Fig. 4.1: Slices of the computed tomography scans of the experimental part in the green state, top row, and after sintering, bottom row. The XY slices between the green state and sintered images are comparable and taken at the crack region of the sintered part.

The primary observation is that the cracks in the middle of the sintered part are not present in the green state. Moreover, the relatively large-scale nature of the defects suggest that they were formed during thermal debinding and perhaps enlarged during sintering. Additionally, we note that both the green and sintered XZ and YZ slices contain a grid-like array of small voids. These voids correspond to the interfilament regions created when cylindrical strands are packed together, forming channels of varying diameter that travel in and out of the XZ and YZ slices. These channels, along with the cross-hatched layering pattern of the pressure map in Figure 3.1a, can also be readily observed in the XY slices of the part. Interestingly, the misshapen discolorations surrounding channels in the damaged XY slice, Figure 4.1d, may correspond to rupturing or possible disintegration within channels and strands brought about by the increased internal pressures and temperatures. Detailed views of the large-scale cracks are shown below in Figure 4.2.

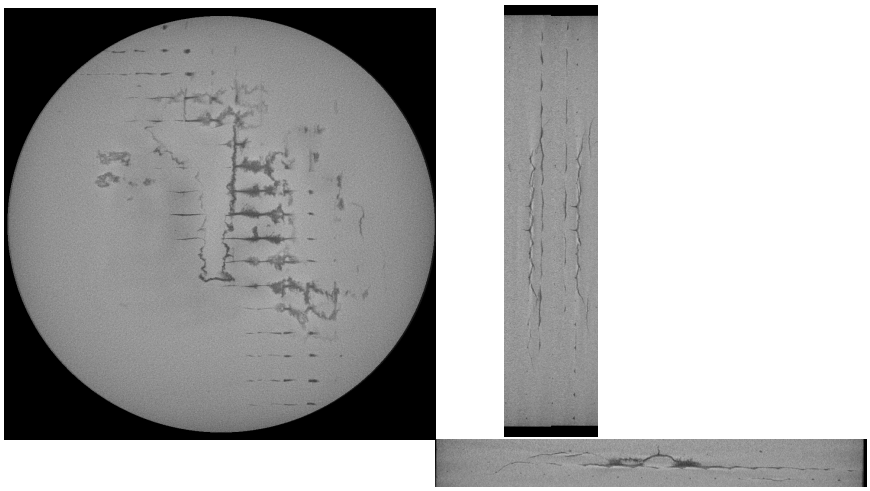


Fig. 4.2: Slices of the computed tomography scans of the damaged region in the sintered part. The XZ slice, bottom, corresponds to a horizontal slice through the XY face. The YZ slices, right and furthest right, correspond to vertical slices through the XY face that cross the ruptured channels at different locations.

A key feature of the observed damage is that the cracks emanate outward from the previously mentioned channels, appearing triangular and aligned with the strands. This is clearly illustrated by the central defect in the XZ slice of Figure 4.2. As this particular defect runs depth-wise into the slice, it carves out a series of cracks that appear almost continuous when viewed from the YZ and XY perspectives. Similar features can be observed in both YZ slices as well. Hence, the experimental images suggest that the larger-scale damage observed in Figure 4.1 is composed of many smaller channel-aligned defects. To simulate this behavior, we embed the the pressure varying material described in Section 3.2 into the strand model as three rods running three parallel channels between strands. The green state implementation of this is shown below in Figure 4.3.

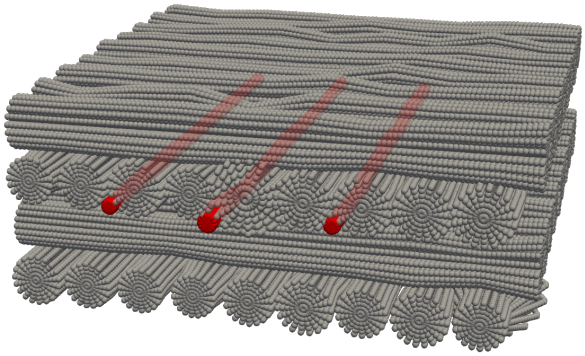
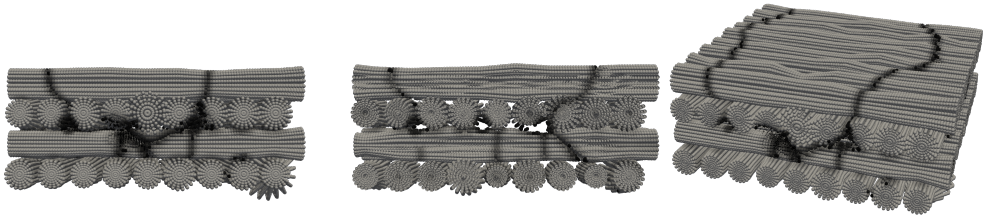


Fig. 4.3: The strand model corresponding to the a subset of the pressure map in the damaged region is shown in gray. The transparent red rods correspond to the pressure material of the thermal debinding model and run the indicated channels.

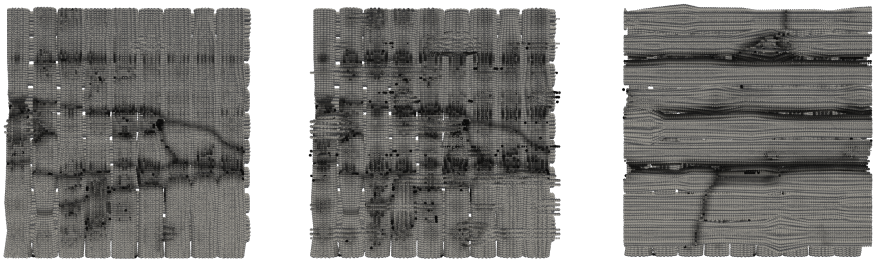
As discussed in Section 3.1, the variations in the strand shape are due to the choice of β and the syringe forces in the vicinity of the damaged region. The result of initializing the thermal debinding simulation with this geometry is presented in Figure 4.4.



(a) (b) (c)

Fig. 4.4: Different views of the damaged part after thermal debinding with darker colors indicating nodes with higher simulated damage, as quantified by (2.14). Nodes with damage in excess of 0.95 have been filtered out to reflect a complete material failure (practically no intact bonds remain). The upper right image corresponds to a slice taken depthwise into Figure 4.4a.

The simulated results contain a number of interesting features. For example, the damage in the channels retain similar characteristics to those observed in the CT scans. The cracks emanate outwards from the channels and travel throughout the part. Adjacent cracks may merge to create larger defects, as can be seen in the central defect of Figure 4.4a. A closer inspection of the channels is provided in Figure 4.5, which displays several XY slices at different heights in the simulated part.



(a) (b) (c)

Fig. 4.5: XY slices of the simulated part after thermal debinding. The slices are ordered in increasing height. Note the pressure material from Figure 4.3 (not shown) runs parallel to the horizontal axis.

In these images, the gaps between strands are clearly visible and comparable to Figure 4.1d and Figure 4.2. Similarly, we observe that the damage is not localized to just the channels, but extends to and through neighboring strands in complex patterns.

5. Conclusion. The present work has demonstrated that peridynamics offers an effective and flexible framework for modeling the direct write additive manufacturing process. The proposed model spans several key components of direct write. First, the green state is assembled by combining a linear strand model with a pressure map detailing the laydown path and syringe forces of the printing procedure. Second, the elevated temperatures and internal pressures associated with thermal debinding are emulated by applying a temperature loading and placing a pressure-varying peridynamic material among the slurry strands of the green state. Third, sintering of the resulting powder compact can carried out by implementing the multiscale peridynamic model developed in [16]. By focusing on the green state and thermal debinding stages of the proposed model, we show that the peridynamic approach capably reproduces key features of defects observed in experiment and appears to show a general qualitative agreement with the available data. It is believed that a more careful selection of model parameters and full implementation of the sintering code would lead closer agreement. To that end, the results of this study motivate a more rigorous and quantitative assessment of the model's validity.

Acknowledgement. The authors would like to thank Stewart Silling for helpful discussions regarding peridynamics and sintering. ASH was supported by funding from the Department of Energy, National Nuclear Security Administration under Award Number DE-NA0002375.

REFERENCES

[1] A. Agwai, I. Guven, and E. Madenci, *Predicting crack propagation with peridynamics: a comparative study*, Int. J. Fract., 171 (2011), pp. 65–78.
[2] E. Askari, F. Bobaru, R. B. Lehoucq, M. L. P. S. A. Silling, and O. Weckner, *Peridynamics for multiscale materials modeling*, Journal of Physics, 125 (2008).
[3] F. Bobaru and M. Duangpanya, *The peridynamic formulation for transient heat conduction*, International Journal of Heat and Mass Transfer, 53 (2010), pp. 4047–4059.
[4] M. Braginsky, V. Tikare, and E. Olevsky, *Numerical simulation of solid state sintering*, Internation Journal of Solids and Structures 42, 42 (2004), pp. 621–636.

[5] P. N. Demmie and S. A. Silling, *An approach to modeling extreme loading of structures using peridynamics*, Journal of Mechanics of Materials and Structures, 2 (2007), pp. 1921–1945.
[6] R. K. Enneti, S. J. Park, R. M. German, and S. V. Atre, *Review: Thermal debinding process in particulate materials processing*, Materials and Manufacturing Processes, 27 (2012), pp. 103–118.
[7] R. M. German, *Thermodynamics of sintering*, in Sintering of Advanced Materials, Z. Z. Fang, ed., Woodhead Publishing, 2010.
[8] I. Gibson, D. W. Rosen, and B. Stucker, *Additive Manufacturing Technologies: Rapid Prototyping to Direct Digital Manufacturing*, Springer Science+Business Media, LLC, 2010.
[9] Y. D. Ha and F. Bobaru, *Studies of dynamic crack propagation and crack branching with peridynamics*, International Journal of Fracture, 162 (2010), pp. 229–244.
[10] K. K. B. Hon, L. Li, and I. M. Hutchings, *Direct writing technology - advances and developments*, CIRP Annals - Manufacturing Technology, 57 (2008), pp. 601–620.
[11] J. A. Lewis, J. E. Smay, J. Stuecker, and J. C. III, *Direct ink writing of three-dimensional ceramic structures*, J. Am. Ceram. Soc., 89 (2006), pp. 3599–3609.
[12] D. J. Littlewood. Private Communication, 6-19-2018.
[13] S. Oterkus, E. Madenci, and A. Agwai, *Peridynamic thermal diffusion*, Journal of Computational Physics, 265 (2014), pp. 71–96.
[14] M. L. Parks, D. J. Littlewood, J. A. Mitchell, and S. A. Silling, *Peridigm users' guide*, Tech. Report SAND2012-7800, Sandia National Laboratories, (2012).
[15] B. C. Salzbrenner, J. M. Rodelas, J. D. Madison, B. H. Jared, L. P. Swiler, Y. Shen, and B. L. Boyce, *High-throughput stochastic tensile performance of additively manufactured stainless steel*, Journal of Materials Processing Technology, 241 (2017), pp. 1–12.
[16] S. Silling, F. Abdeljawad, and K. Ford, *Peridynamic theory as a new paradigm for multiscale modeling of sintering*, Tech. Report SAND2017-10313R, Sandia National Laboratories, (2017).
[17] S. A. Silling, *Reformulation of elasticity theory for discontinuities and long range forces*, Journal of the Mechanics and Physics of Solids, 48 (2000), pp. 175–209.
[18] S. A. Silling and E. Askari, *A meshfree method based on the peridynamic model of solid mechanics*, Computers and Structures, 83 (2005), pp. 1526–1535.
[19] S. A. Silling, M. Epton, O. Weckner, J. Xu, and E. Askari, *Peridynamic states and constitutive modeling*, Journal of Elasticity, 88 (2007), pp. 151–184.
[20] S. A. Silling and R. B. Lehoucq, *Peridynamic theory of solid mechanics*, Adv. Appl. Mech., 44 (2010), pp. 73–168.
[21] S. A. Silling, M. Parks, J. R. Kamm, O. Weckner, and M. Rassaian, *Modeling shockwaves and impact phenomena with eulerian peridynamics*, International Journal of Impact Engineering, 107 (2017), pp. 47–57.
[22] D. Tsai, *Pressure buildup and internal stresses during binder burnout: numerical analysis*, AIChE Journal, 37 (1991), pp. 547–554.
[23] D. Z. Turner, *A non-local model for fluid-structure interaction with applications in hydraulic fracturing*, International Journal for Computational Methods in Engineering Science and Mechanics, 14 (2013), pp. 391–400.
[24] D. Z. Turner, B. G. van Bloemen Waanders, and M. L. Parks, *Inverse problems in heterogeneous and fractured media using peridynamics*, J. Mechanics of Materials and Structures, 10 (2015), pp. 573–590.
[25] J. Xu, E. Askari, O. Weckner, and S. Silling, *Peridynamic analysis of impact damage in composite laminates*, Journal of Aerospace Engineering, 21 (2008), pp. 187–194.

SIMULATION-BASED PARAMETER ESTIMATION APPLIED TO ANTENNA ANALYSIS

LOGAN T. MEREDITH*, DUNCAN A. O. MCGREGOR†, AND RICHARD M. J. KRAMER‡

Abstract. Simulation-based parameter estimation is useful for fitting models to complex physical systems where analytic study is impossible and experimental investigation is impractical. The characterization of antennae frequently fits this description. We describe a method for determining the convolution kernel of an antenna using electromagnetic simulation. The principles described herein are applicable to the analysis of any system that can be described by a linear analog filter.

1. Introduction. Antenna analysis involves measuring the response of antennae to signals of various frequencies. Knowledge of the frequency response of an antenna is vital to understanding its applicability in a given frequency range. This is because antennae have only a finite bandwidth within which they pass a significant signal.

The objective of this paper is to develop a method for measuring the frequency response of antennae using electromagnetic simulation. Ultimately, we wish to use a nonlinear least-squares method to fit the measurements to an easily realizable mathematical model. This reduced order model can make the prediction of the antenna's response to any signal far simpler than with brute force simulation or experimentation.

The method we use to estimate the frequency response of an antenna is similar to that used by scalar network analyzers. These devices are typically comprised of two main components: a spectrum analyzer and a signal generator. The signal generator sweeps through the frequency range of interest, applying a a known input signal. The spectrum analyzer then measures the output signal as a function of frequency. The ratio of the output amplitude to input amplitude is the gain of the network. Most modern spectrum analyzers measure the frequency domain output signal by first measuring the signal in the time domain and then performing a discrete Fourier transform (DFT) [6]. Our approach emulates this technique.

2. Mathematical motivation. In this section we provide mathematical background that will inform our numerical analysis. In general, we use the convention that frequencies are angular, rather than ordinary, and that $j \equiv \sqrt{-1}$.

2.1. Network synthetic approach. We model antennas as linear analog filters. Hence, given a time-varying input signal $x(t)$, the antenna produces a time-varying output signal $y(t)$ according to

$$y(t) = (x * h)(t), \tag{2.1}$$

where h is called the linear response function or, more generally, the convolution kernel of the filter, and $*$ is the convolution operator defined as

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(t-s)g(s)\,ds. \tag{2.2}$$

We shall use the convention that the Fourier transform of a function $f \in L^1(\mathbb{R})$ is defined as

$$\hat{f}(\omega) \equiv \int_{-\infty}^{\infty} f(t)e^{-j\omega t}\,dt. \tag{2.3}$$

*University of Illinois at Urbana-Champaign, logantm2@illinois.edu
†Sandia National Laboratories, damcgre@sandia.gov
‡Sandia National Laboratories, rmkrame@sandia.gov

By taking the Fourier transform of both sides of (2.1) and using the convolution theorem, we find

$$\hat{y}(\omega) = \hat{h}(\omega)\hat{x}(\omega). \tag{2.4}$$

The Fourier transform of the convolution kernel $\hat{h}$ is referred to as the transfer function of the filter.

The transfer function is generally a complex valued function. We refer to the magnitude of the transfer function $\hat{g}(\omega) = |\hat{h}(\omega)|$ as the gain of the filter. This is the definition of gain we use throughout this paper. There is a similar term in the field of antenna analysis called power gain, which involves the directivity and power transmission efficiency of the antenna, and these two terms should not be confused.

The operation of a filter is governed by its convolution kernel. If a filter's transfer function is known, its convolution kernel can be recovered via inverse Fourier transform, given by

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)e^{j\omega t}\, d\omega. \tag{2.5}$$

Note that a signal can be any time dependent quantity. Common signals are voltages, currents, and electric fields in the realm of electrical network analysis.

2.2. Filter fitting methodology. Antennae essentially act as superpositions of bandpass filters. We model the antenna with Butterworth filters [3]. The transfer function of these filters is given by

$$\hat{h}(\omega) = \frac{h_0}{B_n\left(\frac{j\nu_0}{\Delta\nu}\left(\frac{\omega}{\nu_0} - \frac{\nu_0}{\omega}\right)\right)}, \tag{2.6}$$

where $\Delta\nu = \nu_R - \nu_L$ is the filter bandwidth, $\nu_0 = \sqrt{\nu_R\nu_L}$ is the resonant frequency, ν_L and ν_R are the left and right bandpass cutoff frequencies, and B_n is the Butterworth polynomial of integer order n, given by

$$B_n(s) = \begin{cases} \prod_{k=1}^{\frac{n}{2}} \left[s^2 - 2s\cos\left(\pi\frac{2k+n-1}{2n}\right) + 1\right] & n \text{ even}, \\ (s+1)\prod_{k=1}^{\frac{n-1}{2}} \left[s^2 - 2s\cos\left(\pi\frac{2k+n-1}{2n}\right) + 1\right] & n \text{ odd}. \end{cases} \tag{2.7}$$

The polynomial order n is interpreted as the number of elements required to construct an equivalent passive filter in the Cauer topology [4,5]. From (2.6), we can compute the filter's gain as

$$\hat{g}(\omega) \equiv |\hat{h}(\omega)| = \frac{|h_0|}{\sqrt{1 + \left[\frac{\nu_0}{\Delta\nu}\left(\frac{\omega}{\nu_0} - \frac{\nu_0}{\omega}\right)\right]^{2n}}}. \tag{2.8}$$

The transfer function of an antenna is considerably more complicated than a single Butterworth bandpass filter. Hence we approximate the antenna's transfer function with a linear superposition of Butterworth bandpass filter. This superposed transfer function is given by

$$\hat{h}_{\mathbf{p}}(\omega) = \sum_{m=0}^{M-1} \frac{h_{0,m}}{B_{n_m}\left(\frac{j\nu_{0,m}}{\Delta\nu_m}\left(\frac{\omega}{\nu_{0,m}} - \frac{\nu_{0,m}}{\omega}\right)\right)}, \tag{2.9}$$

where M is the number of summed Butterworth bandpass filters and

$$\mathbf{p} = (M, h_{0,0}, \nu_{L,0}, \nu_{R,0}, n_0, \ldots, h_{0,M-1}, \nu_{L,M-1}, \nu_{R,M-1}, n_{M-1})$$

is a vector parametrizing the transfer function. In general,

$$\mathcal{P} = \bigcup_{M=1}^{\infty} \{M\} \times \left(\mathbb{C} \times \mathbb{R}^2 \times \mathbb{Z}_+\right)^M$$

is the space of admissible parameters. Note that the number of filters is itself parametrized. A treatment of this extra complexity is described in Section 3.5. The gain of this filter is not generally expressible analytically, but can be easily numerically computed as

$$\hat{g}_{\mathbf{p}}(\omega) \equiv \left|\hat{h}_{\mathbf{p}}(\omega)\right| = \left|\sum_{m=0}^{M-1} \frac{h_{0,m}}{B_{n_m}\left(\frac{j\nu_{0,m}}{\Delta\nu_m}\left(\frac{\omega}{\nu_{0,m}} - \frac{\nu_{0,m}}{\omega}\right)\right)}\right|. \tag{2.10}$$

The measurement of the convolution kernel of the antenna proceeds as follows. First, we apply an input signal on the antenna in the form of an electromagnetic plane wave in an anechoic chamber. Then, the resulting voltage difference at the coaxial port of the antenna is transformed via discrete Fourier transform (DFT). The ratio of the transforms of the voltage to the input signal is the gain of the antenna. Finally, we fit the gain versus frequency to (2.10). The transfer function and convolution kernel of the antenna can be recovered from the fitted parameters.

3. Problem description. In this section we describe the construction of the simulation, its computational domain, and the boundary conditions with which the problem is driven.

3.1. Domain. The antenna we are testing in this paper is a model SAS-545 biconical antenna from A.H. Systems, Inc. [1]. Such an antenna consists of two solid right conical frusta made of a conducting material whose tops face each other and are separated by a fixed distance [13]. These elements are held in place by a case made of a dielectric between the two elements. Extending from each element are six conducting arms that approximate a larger cone. A diagram of the setup is shown in Figure 3.1.

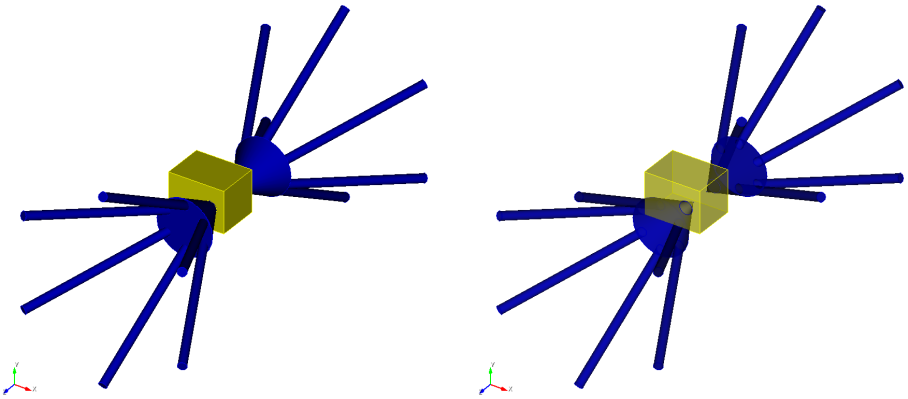
Let Ω_{poles} be the region occupied by the conductive elements of the antenna. Then define

$$\Omega_{\text{TF}} = \left(\left[-\frac{w}{2}, \frac{w}{2}\right]^2 \times \left[-\frac{h}{2}, \frac{h}{2}\right]\right) \setminus \Omega_{\text{poles}} \tag{3.1}$$

as the total field region, where $w, h \in \mathbb{R}_+$ are the width and height of the region, respectively. Further define

$$\Omega_{\text{SF}} = \left(\left[-d - \frac{w}{2}, d + \frac{w}{2}\right]^2 \times \left[-d - \frac{h}{2}, d + \frac{h}{2}\right]\right) \setminus (\Omega_{\text{TF}} \cup \Omega_{\text{poles}}) \tag{3.2}$$

as the scattered field region, where $d \in \mathbb{R}_+$ is the thickness of the region. Then the computational domain is given by $\Omega = \Omega_{\text{TF}} \cup \Omega_{\text{SF}}$. Note that the dielectric antenna case is contained in Ω_{TF}. The significance of the total field and scattered field regions is discussed further in Section 3.2. A cross section of the computational domain is shown in Figure 3.2.



(a) A solid view of the biconical antenna. (b) A transparent view of the biconical antenna.

Fig. 3.1: Isometric view of a biconical antenna, both solid and transparent. Note the separation between the two poles in the transparent view.

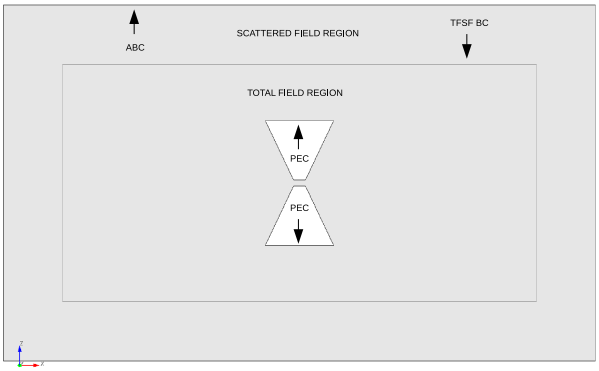


Fig. 3.2: Cross section of the computational domain, showing the regions and imposed boundary conditions. Note that the dielectric antenna case is contained in the total field region. The conductive arms of the antenna are not shown here.

3.2. Drive. For a unique solution to Maxwell's equations within Ω, we must specify boundary conditions on Γ for $0 \leq t \leq t_0$, where t_0 is the termination time of the simulation. Let Γ_{PEC} be the boundary of Ω_{poles}. In the simulation, we impose perfect electrical conductor (PEC) boundary conditions on Γ_{PEC}, since it physically represents the surface of the conductive antenna poles. Now let Γ_{ABC} be the boundary of $[-d - w/2, d + w/2]^2 \times [-d - h/2, d + h/2]$. On Γ_{ABC}, we impose Mur absorbing boundary conditions (ABC) [11], since it physically represents the absorbing walls of an anechoic chamber. Note that $\Gamma = \Gamma_{\text{PEC}} \cup \Gamma_{\text{ABC}}$ is the boundary of Ω. Hence we have imposed boundary conditions completely on the boundary of Ω.

Furthermore, we wish to introduce energy into the domain. We use the total field

scattered field (TFSF) formulation [10, 12] to introduce a plane wave into Ω_{TF}, which impinges upon and scatters off of the antenna. Let $\Gamma_{\mathrm{TFSF}} = \Omega_{\mathrm{TF}} \cap \Omega_{\mathrm{SF}}$ be the TFSF boundary. Then we use the TFSF boundary condition to impose an incident electric field in Ω_{TF} of the form

$$\mathbf{E}_{\mathrm{incident}}(\mathbf{x}, t) = E\left(t - \frac{\hat{\mathbf{k}}}{c} \cdot (\mathbf{x} - \boldsymbol{\phi})\right) \hat{\mathbf{p}}, \tag{3.3}$$

where c is the speed of light, $\hat{\mathbf{k}}$ is the propagation direction, $\boldsymbol{\phi}$ is the phase center, $\hat{\mathbf{p}}$ is the polarization, and $E : \mathbb{R} \mapsto \mathbb{R}$ is an arbitrary amplitude. Intuitively, this electric field specification describes a plane wave pulse of amplitude $E(\tau)$, oriented along $\hat{\mathbf{p}}$, and traveling from $\boldsymbol{\phi}$ in the direction $\hat{\mathbf{k}}$. Note that c is material-dependent, given by $c = 1/\sqrt{\mu\epsilon}$. We choose the permittivity to be given by $\epsilon = 2.25\epsilon_0$ within the antenna case and $\epsilon = \epsilon_0$ everywhere else. The permeability is constant everywhere, as $\mu = \mu_0$.

For our simulation, the termination time is given by $t_0 = 1/3 \times 10^{-6}$ s. Furthermore, we choose $\hat{\mathbf{p}} = \hat{\mathbf{z}}$, $\hat{\mathbf{k}} = -\hat{\mathbf{y}}$, and $\boldsymbol{\phi} = (0, w/2, 0)$. These choices orient the incident electric field to maximize the received signal at the antenna, given a biconical antenna's radiation pattern [9]. Our choice of E merits further motivation and will be discussed in Section 3.3.

3.3. Signal generation and detection. Simulation affords us nearly unlimited choices for input signals. To simplify post processing, it is convenient to choose a signal that is easy to work with.

Consider a time-dependent input signal of the form

$$x(t) = Ae^{-\frac{\sigma^2}{2}(t-\lambda)^2}. \tag{3.4}$$

The Fourier transform of this signal is given by

$$\hat{x}(\omega) = A\sqrt{\frac{2\pi}{\sigma^2}} e^{-j\lambda\omega} e^{-\frac{\omega^2}{2\sigma^2}}. \tag{3.5}$$

The gain of a filter driven by the input signal x can therefore be computed as

$$\hat{g}(\omega) = \frac{|\hat{y}(\omega)|}{|A|} \sqrt{\frac{\sigma^2}{2\pi}} e^{\frac{\omega^2}{2\sigma^2}}. \tag{3.6}$$

Note that the gain is independent of λ.

We desire the gain of a filter only in a certain frequency range. Denote this set of frequencies of interest by $\Phi \subset [0, \infty)$. In order to minimize floating point errors, it is preferable to choose the parameters of x such that

$$\frac{\operatorname{diam} \hat{x}(\Phi)}{\sup |\hat{x}(\Phi)|} \gg 0. \tag{3.7}$$

Since the form of x has been chosen such that $\hat{x}$ is a scaled standard Gaussian function, this can be achieved by taking $\sigma \geq \sup \Phi$. Intuitively, the effect of this is to widen the Gaussian $\hat{x}$ such that the scale does not fluctuate wildly across the frequency range of interest. Such fluctuations would introduce errors in the computation of $\hat{g}$.

Implementing this signal in our simulation is straightforward. Choose $E(\tau) = x(\tau)$ as in (3.3). Then the simulation imparts the desired input signal onto the antenna in the form of an incident electric field. For our simulation, our set of frequencies of interest is given by $\Phi = 2\pi \cdot [3 \times 10^4, 1.5 \times 10^9]$ rad/s, corresponding to a range between 30 MHz and

1.5 GHz. Hence we choose $\sigma = 1.5/4 \times 10^9$ s^{-1}. Furthermore, we choose $A = 1$ V/m and $\lambda = t_0/2 = 5/3 \times 10^{-7}$ s so that the signal is quiescent for $0 \leq t \ll \lambda$ and $t_0 \geq t \gg \lambda$.

Measurement of the output signal can be performed by computing the voltage difference between the poles. Therefore let

$$y(t) = \int_\gamma \mathbf{E}(\mathbf{x}, t) \cdot d\mathbf{l}, \tag{3.8}$$

where γ is the line segment between the poles, $\{0\}^2 \times [-l/2, l/2]$, oriented in the positive z-direction. Note that $\mathbf{E}$ here is the total electric field, consisting of the sum of the incident and scattered fields.

3.4. Discretization. It is vital to choose discretization intervals that are fine enough in order resolve effects of frequency components throughout Φ. In general, it is sufficient to choose $\Delta t \leq \frac{1}{2 \sup \Phi}$ and $\Delta x \leq \frac{c}{2 \sup \Phi}$. Hence we choose $\Delta t = 4/3 \times 10^{-11}$ s and a prescribed edge length of $\Delta x = 0.01$ m with a tetrahedral mesh. The mesh was generated using Cubit [2].

3.5. Post processing. From the simulation, we extract two pieces of data: the input and output signals at each time step. Denote these by the sequences $\{x_i\}$ and $\{y_i\}$, respectively. Use a Fast Fourier Transform algorithm [7] to compute the DFT of each sequence, $\{\hat{x}_i\}$ and $\{\hat{y}_i\}$, respectively. Further denote the sequence of time steps by $\{t_i\}$. The DFT sample frequencies can be computed from $\{t_i\}$. Denote the sequence of sample frequencies by $\{\omega_i\}$. Finally, let

$$\{\hat{g}_i\} = \left\{ \left| \frac{\hat{y}_i}{\hat{x}_i} \right| \right\}. \tag{3.9}$$

Then $\{\hat{g}_i\}$ consists of the computed gain of the antenna at the frequencies $\{\omega_i\}$.

To find our reduced order model, we must use a nonlinear least-squares strategy to fit $\hat{g}_{\mathbf{p}}$ as in (2.10) to $\{\hat{g}_i\}$. The fitting problem can be reduced to finding

$$\arg\min_{\mathbf{p} \in \mathcal{P}} \sum_{i=0}^{N-1} \left| \hat{g}_{\mathbf{p}}(\omega_i) - \hat{g}_i \right|^2, \tag{3.10}$$

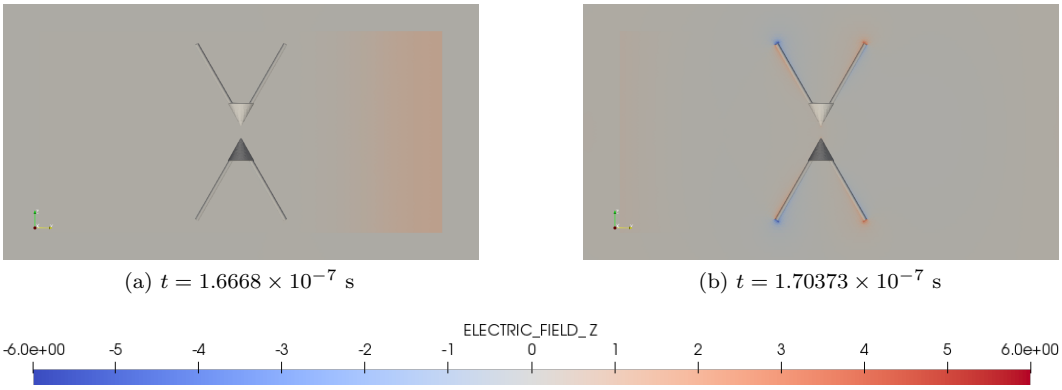
where $\mathcal{P}$ is the space of admissible parameters.

This problem can be further reduced to the subproblem

$$\arg\min_{(M, \bar{\mathbf{p}}) \in \mathcal{P}} \sum_{i=0}^{N-1} \left| \hat{g}_{(M, \bar{\mathbf{p}})}(\omega_i) - \hat{g}_i \right|^2 \tag{3.11}$$

for fixed $M \in \mathbb{Z}_+$, which effectively reduces the fitted function to be composed of M Butterworth filters. By inspection, M cannot exceed the number of local maxima, or peaks, in $\{\hat{g}_i\}$. Thus there is an upper bound on M. In order to find a best fit, we solve (3.11) for all M less than or equal to the number of peaks in $\{\hat{g}_i\}$. For each such M, we then compute the L^2 error norm for the corresponding fit. The fit for M with the least error norm is the best fit.

4. Results. The simulation was performed using the EMPHASIS electromagnetics code [14]. On the Skybridge capacity cluster, it took 12 hours on 368 cores, and contained 14.5 million finite elements. Figure 4.1 shows a cross section of the domain as the Gaussian electric field pulse impacts the antenna at multiple time steps.

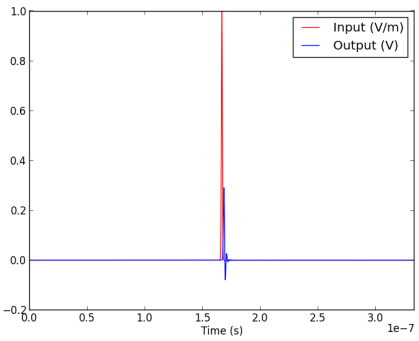


(a) $t = 1.6668 \times 10^{-7}$ s

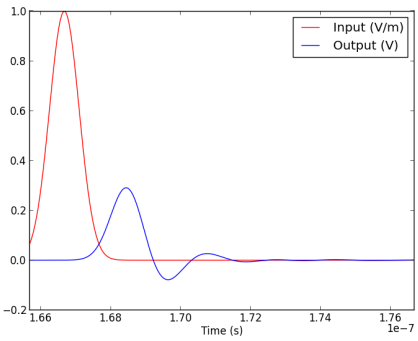
(b) $t = 1.70373 \times 10^{-7}$ s

Fig. 4.1: Cross section of the domain during impingement of the Gaussian electric field pulse on the antenna, showing the z-component of the electric field. The electric field is in units of V/m.

We are especially interested in the voltage difference between the antenna poles, which acts as our output signal. Figure 4.2 displays this voltage difference alongside the input signal at every time step in the simulation. The figure also shows a zoomed view about $t = t_0/2$ so that the narrow Gaussian envelope of the input signal is visible.



(a) Input and output signals as functions of time.

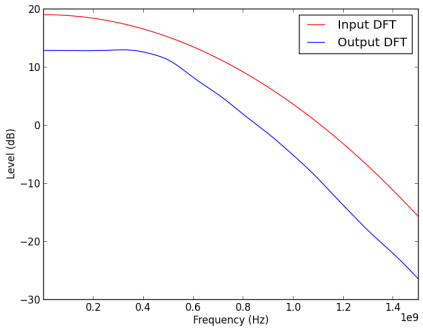


(b) Zoomed view of input and output signals near $t = t_0/2$.

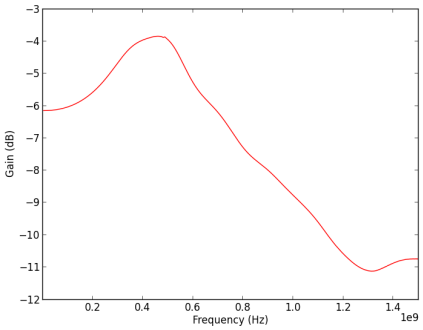
Fig. 4.2: Raw signal data collected from the simulation. The input corresponds to$\{x_i\}$ and the output corresponds to $\{y_i\}$.

To post process the signal data, we must first compute the DFT of both the input and output. Before computing the DFT, we apply a Hamming window to the signals to reduce spectral leakage [8]. Since the input signal approximates a Gaussian, the absolute value of its DFT also approximates a Gaussian. This can be clearly seen in Figure 4.3a, which also displays the DFT of the output signal. Figure 4.3b shows the gain of the antenna, computed according to (3.9).

Finally, we wish to fit a reduced order model to our data. To do so, we follow the procedure outlined in Section 3.5. There are 13 peaks in the computed gain data, found by



(a) Absolute value of DFT of signals.



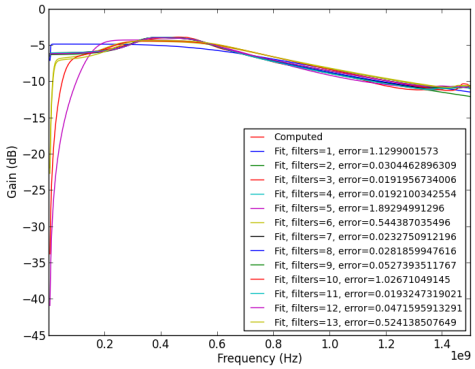
(b) Computed gain of antenna.

Fig. 4.3: DFT of raw signal data collected from the simulation and gain thence computed.

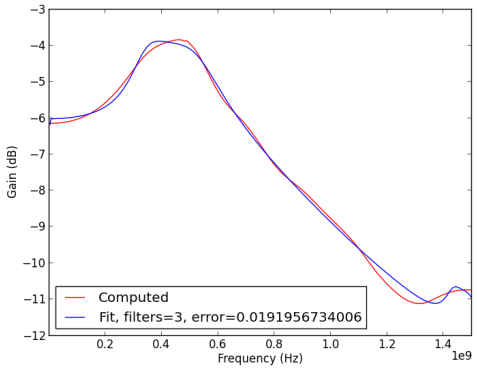
comparing neighboring values. Hence we perform a nonlinear least-squares fit for numbers of filters up to and including 13. Figure 4.4a shows every such fit superimposed on $\{\hat{g}_i\}$, and Figure 4.4b shows only the best fit. We find that an optimally fit Butterworth bandpass filter as in (2.10) has 3 filters and is parametrized by

$$\begin{aligned}\mathbf{p} = (&3, 2.49968544 \times 10^{-1}\ \text{m}, 1.58368052 \times 10^{6}\ \text{Hz}, 7.13738696 \times 10^{8}\ \text{Hz}, 2,\\ &-1.70516259 \times 10^{-1}\ \text{m}, 3.21217315 \times 10^{8}\ \text{Hz}, 5.73665070 \times 10^{8}\ \text{Hz}, 2,\\ &2.11043001 \times 10^{-2}\ \text{m}, 1.51721542 \times 10^{9}\ \text{Hz}, 1.41792554 \times 10^{9}\ \text{Hz}, 2). \quad (4.1)\end{aligned}$$

This fitted Butterworth bandpass filter can be used to predict the response of the antenna to any time-varying electric field according to (2.1).



(a) Absolute value of DFT of signals.



(b) Computed gain of antenna.

Fig. 4.4: DFT of raw signal data collected from the simulation and gain thence computed.

5. Conclusions. It is undesirable to perform a fully resolved electromagnetic simulation for every input signal of interest on an antenna. As we have described, however, it is feasible to capture sufficient data from a single simulation to estimate the parameters of a known

linear analog filter. This permits the prediction of the antenna's response to any arbitrary time-dependent input signal.

In principle, we do not have to restrict our study to antennae. Any linear analog filter design with well defined input and output signals can be treated in an analogous way to the antenna presented here.

REFERENCES

[1] A.H. Systems, Inc., *SAS-545 Biconical Antenna Operation Manual*, Chatsworth, CA, Nov. 2014.

[2] T. D. Blacker, W. J. Bohnhoff, and T. L. Edwards, *Cubit mesh generation environment. volume 1: Users manual*, tech. rep., Sandia National Labs., Albuquerque, NM (United States), 1994.

[3] S. Butterworth, *On the theory of filter amplifiers*, Wireless Engineer, 7 (1930), pp. 536–541.

[4] G. A. Campbell, *Physical theory of the electric wave-filter*, Bell System Technical Journal, 1 (1922), pp. 1–32.

[5] W. Cauer, *Die verwirklichung von wechselstromwiderständen vorgeschriebener frequenzabhängigkeit*, Archiv für Elektrotechnik, 17 (1926), pp. 355–388.

[6] J. Cooley, P. Lewis, and P. Welch, *The finite fourier transform*, IEEE Transactions on audio and electroacoustics, 17 (1969), pp. 77–85.

[7] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex fourier series*, Mathematics of computation, 19 (1965), pp. 297–301.

[8] F. J. Harris, *On the use of windows for harmonic analysis with the discrete fourier transform*, Proceedings of the IEEE, 66 (1978), pp. 51–83.

[9] J. G. Maloney, G. S. Smith, and W. R. Scott, *Accurate computation of the radiation from simple antennas using the finite-difference time-domain method*, IEEE Transactions on Antennas and Propagation, 38 (1990), pp. 1059–1068.

[10] D. Merewether, R. Fisher, and F. Smith, *On implementing a numeric huygen's source scheme in a finite difference program to illuminate scattering bodies*, IEEE Transactions on Nuclear Science, 27 (1980), pp. 1829–1833.

[11] G. Mur, *Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations*, IEEE transactions on Electromagnetic Compatibility, (1981), pp. 377–382.

[12] D. J. Riley, J.-M. Jin, Z. Lou, and L. R. Petersson, *Total-and scattered-field decomposition technique for the finite-element time-domain method*, IEEE transactions on antennas and propagation, 54 (2006), pp. 35–41.

[13] R. Roan and K. A. Zaki, *Application of the fdtd method to the analysis of biconical antennas*, in Antennas and Propagation Society International Symposium, 1998. IEEE, vol. 1, IEEE, 1998, pp. 516–519.

[14] C. D. Turner, D. B. Seidel, and M. F. Pasik, *EMPHASIS/Nevada UTDEM User Guide: Version 1.0*, United States. Department of Energy, 2005.

A NUMERICAL APPROACH TO SIMULATING MAGNETIC FIELDS AND SPIN-ORBIT COUPLING IN QUANTUM DOTS

MITCHELL BRICKSON*, N. TOBIAS JACOBSON†, AND ANDREW D. BACZEWSKI‡

Abstract. Understanding the effects of magnetic fields and spin-orbit coupling (SOC) on the properties of quantum dots (QDs) is essential to creating models with sufficient richness to explore qubit applications *in silico* and to help interpret experiments. In this article we show how to incorporate these effects into an interior penalty discontinuous Galerkin (IPDG) discretization of the Schrödinger-like equations that govern effective mass theory. We describe the implementation of this approach in the Laconic software package and present results that numerically verify it against a variety of test problems. We conclude by describing some of the calculations that our IPDG capability will enable.

1. Introduction. Lithographic quantum dots (QDs) are engineered few-level quantum systems created by patterning nanoscale control electrodes in the vicinity of a semiconductor heterostructure. The band offsets between the materials in the heterostructure and the built-in potential that arises due to background doping lead to the formation of two-dimensional accumulation or inversion layers at material interfaces. By carefully tuning the voltages applied to the control electrodes, the local density of carriers in these layers can be manipulated such that one or a few can be separated from the rest, and then manipulated and measured. Leveraging many years of advances in semiconductor device fabrication technologies, single- or few-carrier QDs have been realized in GaAs/AlGaAs [9], Si/MOS [12], Si/SiGe [7], and Ge/SiGe [3] heterostructures. Such devices show great promise for realizing quantum technologies, particularly quantum information processing devices. While many of these QDs have been used as qubits themselves, they are also used for qubit readout [10] and in heterogeneous two-qubit systems [4].

Impressed magnetic fields and intrinsic spin-orbit coupling (SOC) play an important role in determining the properties of these systems pertinent to their use as qubits. A recent experiment in a Si/MOS double QD illustrated that the combination of a uniform magnetic field and variations in the SOC between QDs can create an effective gradient magnetic field. This gradient field was strong enough driving rotations in a singlet-triplet qubit encoded in the electronic spin state of electrons occupying the double QD [6]. Also of interest are recent proposals for driving qubit rotations via electric-dipole spin resonance, leveraging the strong SOC in Ge quantum wells [11]. To help guide experimental progress, device-level models that can be used to compute the properties of QDs are frequently used. However, to the best of our knowledge there is presently no mature device-level modeling tool that incorporates both magnetic fields and SOC.

In this article, we describe the implementation of magnetic fields and SOC to facilitate device-level modeling of QDs in the Laconic software package. Laconic implements an interior penalty discontinuous Galerkin (IPDG) discretization [5] of the Schrödinger-like equations that govern effective mass theory. The IPDG framework is used to incorporate features of semi-analytic solutions in regions with known physics through the use of enrichment functions. For example, in accurately capturing the wave function in the vicinity of singular potentials (e.g., impurity atoms) or at heterostructure interfaces. This is essential because these difficult to capture features can be disproportionately responsible for the system's energetics or even its fine structure (e.g., valley splitting). Naive discretizations might require prohibitively large linear systems to accurately represent the relevant physics.

*Center for Quantum Information and Control, University of New Mexico, mbrickson@unm.edu
†Center for Computing Research, Sandia National Laboratories, ntjacob@sandia.gov
‡Center for Computing Research, Sandia National Laboratories, adbacze@sandia.gov

In Section 2 we begin by reviewing the Schrödinger-like equations that typically appear in effective mass theory and describe the manner in which magnetic fields and SOC enter into those equations. We next describe the standard IPDG discretization used in Laconic in Section 3 and describe the modifications necessary to incorporate magnetic fields and SOC in Sections 3.1 and 3.2. These descriptions are furnished at a physicist's level of rigor and serve to document the practical details of our implementation. Illustrative numerical results that verify our implementation are include in Section 4. Finally, we summarize our results and consider the types of calculations that this will enable in Section 5.

2. Effective mass theory. The central ansatz in effective mass theory is that the wave function of a localized charge carrier, $\psi(\boldsymbol{x})$, can be factorized into a product of an envelope function, $F(\boldsymbol{x})$, and a Bloch function, $\phi(\boldsymbol{x})$,

$$\psi(\boldsymbol{x}) = F(\boldsymbol{x})\phi(\boldsymbol{x}). \tag{2.1}$$

The envelope function describes the coarse scale behavior of the carrier due to, e.g., confinement by an applied potential, whereas the Bloch function describes the atomic scale behavior of the carrier as it would behave in a bulk crystal. This simple picture assumes that the wave function is primarily supported in the vicinity of a single band extremum, though we note that it is possible to generalize to multi-valley effective mass theories in which this support is distributed across extrema that would be degenerate in a bulk crystal (e.g., in silicon [1, 2]). For simplicity, we consider only the single extremum case in this article.

By factoring the coarse and atomic scale features of the wave function and holding the Bloch function fixed to its bulk functional form, the effects of the applied potential and the bulk crystal's lattice potential can be separated. A Schrödinger-like equation for the envelope function can be derived, taking the form

$$\left(\frac{1}{2}\hat{\boldsymbol{p}} \cdot \boldsymbol{m}_{\mathit{eff}}^{-1} \cdot \hat{\boldsymbol{p}} + V(\boldsymbol{x})\right) F(\boldsymbol{x}) = EF(\boldsymbol{x}), \tag{2.2}$$

where $\hat{\boldsymbol{p}}$ is the canonical momentum operator, $\boldsymbol{m}_{\mathit{eff}}$ is the effective mass tensor associated with the band extremum of interest, and E is the energy relative to that extremum. Given the appearance of (2.2) as a Schrödinger-like equation, one can think of the operator acting on F as an effective Hamiltonian, and we will henceforth refer to it as such. For the sake of simple presentation, the rest of this report will focus on the special case of an isotropic effective mass tensor, indicated by the single scalar m_{eff}. However, the results here are readily generalized to arbitrary effective mass tensors.

To incorporate the effect of a magnetic field into Eq. 2.2, we transform the canonical momentum operator to include the magnetic vector potential, $\boldsymbol{A}$,

$$\hat{\boldsymbol{p}} = -i\hbar\nabla \rightarrow \hat{\boldsymbol{p}} = -i\hbar\nabla + e\boldsymbol{A}, \tag{2.3}$$

where e is the charge of the particle and the magnetic field is related to the vector potential by $\boldsymbol{B} = \nabla \times \boldsymbol{A}$. This augments the kinetic energy, the numerical aspects of which will be one of the focuses of this paper.

When considering the reference frame of a charged particle moving through an electric potential, the motion of the potential relative to the particle can be seen as inducing an effective magnetic field in that frame. This leads to an additive contribution to the bare Hamiltonian referred to as the SOC, which can be expressed as

$$\frac{\hbar}{4m^2c^2}\hat{\boldsymbol{\sigma}} \cdot (\nabla V \times \hat{\boldsymbol{p}})\,. \tag{2.4}$$

Note that the mass appearing in this expression is the bare electronic mass, and renormalization into an effective mass theory is non-trivial and outside the scope of this report. Independent of details pertaining to the incorporation of this physics into effective mass theory, taking advantage of this effect has become a central point of some schemes for controlling qubits in QDs as the presence of SOC allows for fast electrical control of the spin state via electric-dipole spin resonance.

3. IPDG Formulation. Proceeding at a physicist's level of rigor, we begin by considering the standard IPDG formulation of a Schrödinger-like equation in the absence of a magnetic field or SOC [8]. The domain supporting $F(\boldsymbol{x})$ is decomposed into a mesh consisting of N_e elements, $\{D^{(a)}\}_{a=1}^{N_e}$. Local to each element is a set of $N_p(a)$ basis functions, $\{\ell_j^{(a)}(\boldsymbol{x})\}_{j=1,a=1}^{N_p(a),N_e}$. While element-local enrichment using semianalytic solutions is the ultimate goal of our approach, we need not consider these details to establish our formulation. For the sake of simplicity, we henceforth suppress the dependence of N_p on the element index, i.e., each element hosts a local basis of the same order. The results in this report will make use of a uniform nodal basis defined over a tetrahedral mesh, for which the nodal polynomials are defined making use of a modal basis of orthogonal Jacobi polynomials as in [5]. We then expand the envelope function as

$$F(\boldsymbol{x}) = \sum_{a=1}^{N_e} \sum_{j=1}^{N_p} F(\boldsymbol{x}_j^{(a)}) \ell_j^{(a)}(\boldsymbol{x}). \tag{3.1}$$

It is evident in the construction of the element-local basis functions that there is no explicit inter-elemental continuity. Thus, an interior penalty formulation is employed to enforce continuity numerically. Intuitively, we force the low-energy solutions to be continuous by artificially adding energy to solutions with discontinuities.

Because the inclusion of the vector potential and spin-orbit operators are the primary point of this paper, we simply state the matrix elements necessary to translate the eigenproblem in (2.2) using the basis in (3.1) into a finite-dimensional generalized eigenproblem. We first define the matrices

$$\mathcal{S}_{jk}^{\mu,(a)} = \int_{D^{(a)}} \mathrm{d}^3\mathrm{x} \ell_j^{(a)} \frac{\partial}{\partial \mu} \ell_k^{(a)}, \quad \mathcal{N}_{jk}^{\mu,(a,b)} = \int_{\partial D^{(a)} \cap \partial D^{(b)}} \mathrm{d}^2\mathrm{x} \ell_j^{(a)} \hat{n}^{(a)} \cdot \nabla \ell_k^{(b)},$$
$$\mathcal{M}_{jk}^{(a)} = \int_{D^{(a)}} \mathrm{d}^3\mathrm{x} \ell_j^{(a)} \ell_k^{(a)}, \quad \mathcal{B}_{jk}^{\mu,(a,b)} = \int_{\partial D^{(a)} \cap \partial D^{(b)}} \mathrm{d}^2\mathrm{x} \hat{\mu} \cdot \hat{n}^{(a)} \ell_j^{(a)} \ell_k^{(b)},$$
$$\text{and } \mathcal{V}_{jk}^{(a)} = \int_{D^{(a)}} \mathrm{d}^3\mathrm{x} \ell_j^{(a)} V \ell_k^{(a)},$$

where μ is any Cartesian coordinate and $\hat{n}^{(a)}$ is the surface normal vector for mesh element a. The matrices with one superscript will appear in blocks of Hamiltonian matrix elements that arise due to basis functions with support over the volume of a single element, whereas the matrices with two superscripts will appear in blocks of Hamiltonian matrix elements that arise due to basis functions with support at the interface between two elements. The diagonal blocks of the Hamiltonian matrix will then take the form

$$\mathcal{H}^{(a,a)} = \frac{\hbar^2}{2m_{\mathit{eff}}} \sum_{\mu \in C} \left(\mathcal{S}^{\mu,(a)}\right)^T \left(\mathcal{M}^{(a)}\right)^{-1} \mathcal{S}^{\mu,(a)}$$
$$- \frac{\hbar^2}{4m_{\mathit{eff}}} \sum_{b \neq a} \left(\mathcal{B}^{(a,b)} \mathcal{N}^{(a,b)} + \left(\mathcal{N}^{(a,b)}\right)^T \mathcal{B}^{(a,b)} - \frac{\alpha}{2} \mathcal{B}^{(a,a)} \right) + \mathcal{V}^{(a)}, \tag{3.2}$$

while the off-diagonal blocks take the form

$$\mathcal{H}^{(a,b)} = \frac{\hbar^2}{4m_{\mathit{eff}}}\left(\left(\mathcal{N}^{(a,b)}\right)^T \mathcal{B}^{(a,b)} - \mathcal{B}^{(a,b)}\mathcal{N}^{(b,a)} + \frac{\alpha}{2}\mathcal{B}^{(a,b)}\right). \tag{3.3}$$

The isolated $\mathcal{B}^{(a,b)}$ matrices in these equations do not correspond to any operators in the original Schrödinger equation, but rather to the interior penalty, and α is the penalty strength. Because our basis is not orthonormal, we note that the right-hand side of the matrix form of (2.2) will include a block-diagonal overlap matrix consisting of entries of $\mathcal{M}^{(a)}$. Due to the structure of this matrix, its inverse can be efficiently applied to $\mathcal{H}$ to yield a standard eigenproblem.

3.1. IPDG with magnetic fields. Having reviewed the matrix elements necessary for the standard IPDG formulation, we next determine the additive contribution to the Hamiltonian matrix that arises due to the transformation of the momentum operator to include a vector potential (see (2.3)). It is convenient to define an intermediate function, $\boldsymbol{\gamma}(\boldsymbol{x})$, the momentum operator acting on the envelope function,

$$\hat{\boldsymbol{p}}F(\boldsymbol{x}) = \boldsymbol{\gamma}(\boldsymbol{x}). \tag{3.4}$$

Then we define a residual of the form $\hat{\boldsymbol{p}}F(\boldsymbol{x}) - \boldsymbol{\gamma}(\boldsymbol{x})$ on which we impose orthogonality to any arbitrary vector test function $\boldsymbol{\nu}$. After some manipulation we can arrive at an elemental strong form of this residual,

$$\int_{D^{(a)}} \mathrm{d}^3\mathrm{x}\boldsymbol{\nu}\cdot\left(-i\hbar\nabla F^{(a)} + e\boldsymbol{A}F^{(a)} - \boldsymbol{\gamma}^{(a)}\right) = -i\hbar\oint_{\partial D^{(a)}} \mathrm{d}^2\mathrm{x}\boldsymbol{\nu}\cdot\hat{n}^{(a)}\left(F^{(a)} - \tilde{F}\right) \tag{3.5}$$

where functions with a superscript (a) are the restriction of that function to mesh element a, $\hat{n}^{(a)}$ is the normal pointing out of mesh element a, and $\tilde{F}$ is the numerical flux, for which we will use a central flux in this situation.

Consider expansions of F and $\boldsymbol{\gamma}$ in terms of our basis functions, $F^{(a)}(\boldsymbol{x}) = \sum_j F(\boldsymbol{x}_j^{(a)})\ell_j^{(a)}(\boldsymbol{x})$ and $\boldsymbol{\gamma}^{(a)} = \sum_{\mu\in C}\hat{\mu}\sum_j \gamma_j^{\mu,(a)}\ell_j^{(a)}$, where $C = \{x, y, z\}$. The orthogonality of the test function to the residual can be imposed if the residual is orthogonal to each basis function. If we define the matrix

$$\mathcal{A}_{jk}^{\mu,(a)} = \int_{D^{(a)}} \mathrm{d}^3\mathrm{x}\ell_j^{(a)}A_\mu\ell_k^{(a)},$$

then, after considering that the central flux is defined as the average value of a function on either side of a boundary, we get the matrix equation

$$\left(-i\hbar\mathcal{S}^{\mu,(a)} + e\mathcal{A}^{\mu,(a)} + \frac{i\hbar}{2}\mathcal{B}^{\mu,(a,a)}\right)F^{(a)} - \frac{i\hbar}{2}\sum_{b\neq a}\mathcal{B}^{\mu,(a,b)}F^{(b)} = \mathcal{M}^{(a)}\boldsymbol{\gamma}^{\mu,(a)}. \tag{3.6}$$

Of course, it is $\hat{\boldsymbol{p}}^2$ rather than $\hat{\boldsymbol{p}}$ that enters into the Hamiltonian in (2.2), the discretization of which was considered in Section 3. To incorporate the transformed vector potential into that formulation, we define a residual and impose orthogonality to test functions just as before, and we use a flux that will include interior penalty terms to increase the energy of discontinuous solutions. The form of this flux is

$$\tilde{\gamma}^\mu = \{\{\gamma^\mu\}\} - \alpha\hat{\mu}\cdot\hat{n}^{(a)}[\![F]\!] \tag{3.7}$$

where $\{\{\gamma^\mu\}\}$ is the average of γ^μ across a boundary, and $[\![F]\!]$ is the jump in F across a boundary. If we define $\sum_{\mu\in C}\mathcal{B}^{\mu,(a,b)}=\mathcal{B}^{(a,b)}$, then this leads to the matrix equation in IPDG form for (2.2)

$$\frac{1}{2m_{\mathit{eff}}}\sum_{\mu\in C}\left(\left(-i\hbar\mathcal{S}^{\mu,(a)}+e\mathcal{A}^{\mu,(a)}+\frac{i\hbar}{2}\mathcal{B}^{\mu,(a,a)}\right)\boldsymbol{\gamma}^{\mu,(a)}-\frac{i\hbar}{2}\sum_{b\neq a}\mathcal{B}^{\mu,(a,b)}\boldsymbol{\gamma}^{\mu,(b)}\right)$$
$$=\left(E\mathcal{M}^{(a)}+\frac{i\hbar\alpha}{4m_{\mathit{eff}}}\mathcal{B}^{(a,a)}-\mathcal{V}^{(a)}\right)F^{(a)}-\frac{i\hbar\alpha}{4m_{\mathit{eff}}}\sum_{b\neq a}\mathcal{B}^{(a,b)}F^{(b)}. \quad (3.8)$$

Putting equations (3.6) and (3.8) together, we can find the contribution that the vector potential makes to the kinetic energy in the Hamiltonian. Giving each block of this addition the variable $\mathcal{C}_{\mathrm{vp}}^{(a,b)}$, for the diagonal blocks we get

$$\mathcal{C}_{\mathrm{vp}}^{(a,a)}=\frac{1}{2m_{\mathit{eff}}}\sum_{\mu\in C}\Bigg(-i\hbar e\left(\mathcal{S}^{\mu,(a)}-\frac{1}{2}\mathcal{B}^{\mu,(a,a)}\right)\left(\mathcal{M}^{(a)}\right)^{-1}\mathcal{A}^{\mu,(a)}-$$
$$i\hbar e\mathcal{A}^{\mu,(a)}\left(\mathcal{M}^{(a)}\right)^{-1}\left(\mathcal{S}^{\mu,(a)}-\frac{1}{2}\mathcal{B}^{\mu,(a,a)}\right)+e^2\mathcal{A}^{\mu,(a)}\left(\mathcal{M}^{(a)}\right)^{-1}\mathcal{A}^{\mu,(a)}\Bigg) \quad (3.9)$$

and for the off-diagonal blocks we get

$$\mathcal{C}_{\mathrm{vp}}^{(a,b)}=-\frac{i\hbar e}{4m_{\mathit{eff}}}\sum_{\mu\in C}\left(\mathcal{A}^{\mu,(a)}\left(\mathcal{M}^{(a)}\right)^{-1}\mathcal{B}^{\mu,(a,b)}+\mathcal{B}^{\mu,(a,b)}\left(\mathcal{M}^{(b)}\right)^{-1}\mathcal{A}^{\mu,(b)}\right). \quad (3.10)$$

The $\mathcal{A}^{\mu,(a)}\left(\mathcal{M}^{(a)}\right)^{-1}\mathcal{A}^{\mu,(a)}$ term in the equation for the diagonal elements, which corresponds to the $\boldsymbol{A}^2$ term of $\hat{\boldsymbol{p}}^2$, can be handled in Laconic in the same manner as a scalar potential. The rest, which corresponds to the $\boldsymbol{A}\cdot\nabla+\nabla\cdot\boldsymbol{A}$ term in the Hamiltonian, was the first addition we made to Laconic.

3.2. IPDG with SOC. For the spin-orbit operator, we only need to focus on the spatial part of the operator corresponding to $\nabla V\times\hat{\boldsymbol{p}}$, since the tensor products with spin degrees of freedom will be trivial to add in after that. Starting with the x-component of this operator we take the same approach as before. We first define α^x as the transformation of F under this operator, then again define a residual and impose orthogonality to test functions ϕ to arrive at

$$\int_{D^{(a)}}\mathrm{d}^3\mathrm{x}\phi\left(-i\hbar\frac{\partial V}{\partial y}\frac{\partial}{\partial z}+i\hbar\frac{\partial V}{\partial z}\frac{\partial}{\partial y}+e\frac{\partial V}{\partial y}A_z-e\frac{\partial V}{\partial z}A_y\right)F^{(a)}-\int_{D^{(a)}}\mathrm{d}^3\mathrm{x}\phi\alpha^{x,(a)}$$
$$=-i\hbar\oint_{\partial D^{(a)}}\mathrm{d}^2\mathrm{x}\phi\left(\frac{\partial V}{\partial y}-\frac{\partial V}{\partial z}\right)\left(F^{(a)}-\tilde{F}\right). \quad (3.11)$$

This can be used to directly determine the matrix equations for the IPDG form of this operator. Using a central flux for $\tilde{F}$ and then defining the matrices

$$\mathcal{E}_{jk}^{\lambda,(a)}=\int_{D^{(a)}}\mathrm{d}^3\mathrm{x}\ell_j^{(a)}\frac{\partial V}{\partial\lambda}\ell_k^{(a)},\quad \mathcal{F}_{jk}^{\lambda,(a)}=\sum_{\mu\in C}\sum_{\nu\in C}\epsilon_{\lambda\mu\nu}\int_{D^{(a)}}\mathrm{d}^3\mathrm{x}\ell_j^{(a)}\frac{\partial V}{\partial\mu}A_\nu\ell_k^{(a)},$$
$$\text{and}\quad \mathcal{J}_{jk}^{\mu\nu,(a,b)}=\int_{\partial D^{(a)}\cap\partial D^{(b)}}\mathrm{d}^2\mathrm{x}\ell_j^{(a)}\frac{\partial V}{\partial\mu}\hat{\nu}\cdot\hat{n}^{(a)}\ell_k^{(b)}$$

where $\epsilon_{\lambda\mu\nu}$ is the standard Levi-Civita symbol, we can then generalize to determine that the spatial operator that goes with $\hat{\sigma}_\lambda$ will have diagonal blocks defined by

$$\mathcal{C}_{\text{so}}^{\lambda,(a,a)} = -i\hbar \sum_{\mu\in C}\sum_{\nu\in C} \epsilon_{\lambda\mu\nu}\left(\mathcal{E}^{\mu,(a)}\left(\mathcal{M}^{(a)}\right)^{-1}\mathcal{S}^{\nu,(a)} - \frac{1}{2}\mathcal{J}^{\mu\nu,(a,a)}\right) + e\mathcal{F}^{\lambda,(a)} \tag{3.12}$$

and off-diagonal blocks defined by

$$\mathcal{C}_{\text{so}}^{\lambda,(a,b)} = -\frac{i\hbar}{2}\sum_{b\neq a}\sum_{\mu\in C}\sum_{\nu\in C}\epsilon_{\lambda\mu\nu}\mathcal{J}^{\mu\nu,(a,b)}. \tag{3.13}$$

4. Results. We have extended the Laconic software package to include implementations of our formulations of the magnetic field and SOC terms. To verify their functionality, we focus on two generic approaches to testing both of them, and a third specific to the magnetic field. The first approach constructs matrix discretizations of these terms, applies them to vectors containing nodal data sampled from known functions, and checks that the resulting nodal data approximately reproduce analytic results. Our second approach is to add these matrices as perturbations to unperturbed Hamiltonians with analytic solutions, and to check that the resulting eigenvectors approximately agree with analytic results obtained with first order perturbation theory. Finally, for the magnetic field term we also verify our numerical solutions against an analytically solvable problem. Respectively, we refer to these approaches as the operator, perturbative, and analytic tests.

4.1. Operator tests. In the same way that ∇^2 can transform an arbitrary function, our magnetic field and SOC operators can be seen abstractly as operators that transform functions. This gives us an especially simple way to do a first test on the operators we have developed here: apply them to a DG representation of a known analytic function and see if the result matches what we would expect to have by doing the calculation analytically. Starting with the function

$$f(x,y,z) = \prod_{\mu\in C}\cos(\pi\mu), \tag{4.1}$$

we can apply $\boldsymbol{A}\cdot\nabla + \nabla\cdot\boldsymbol{A}$ resulting in

$$(\boldsymbol{A}\cdot\nabla + \nabla\cdot\boldsymbol{A})\, f = \sum_{\mu\in C}\left(-2\pi A_\mu \sin(\pi\mu) + \frac{\partial A_\mu}{\partial\mu}\cos(\pi\mu)\right)\prod_{\nu\neq\mu}\cos(\pi\nu). \tag{4.2}$$

For a fixed mesh and an associated nodal basis, we can perform a Gram test to generate a representation of f to which the matrix form of $\boldsymbol{A}\cdot\nabla+\nabla\cdot\boldsymbol{A}$ can be applied. The result of this matrix-vector multiplication is then another vector containing an approximate representation of (4.2), that can be compared to the analytic result. For all tests in this Section, we use a tetrahedral mesh of a cubic domain with edge length 1 and typically elemental edge lengths of 0.1. Our tests indicate that our implementation of the magnetic field operator produces approximate results that agree well with the analytic calculation and that increasing the order of the nodal basis uniformly decreases the error in the approximation to (4.2). Exemplary results for our operator tests can be found in Figures 4.1 and 4.2.

To test our formulation of the SOC operator, we continue to ignore the spin degree of freedom and focus on the spatial part of the operator. In units with $\hbar = e = 1$, applying $(\nabla V\times\hat{\boldsymbol{p}})_x$ to f will result in

$$(\nabla V\times\hat{\boldsymbol{p}})_x\, f = \frac{\partial V}{\partial y}\left(i\pi\sin(\pi z) + A_z\cos(\pi z)\right)\prod_{\mu\neq z}\cos(\pi\mu) \tag{4.3}$$

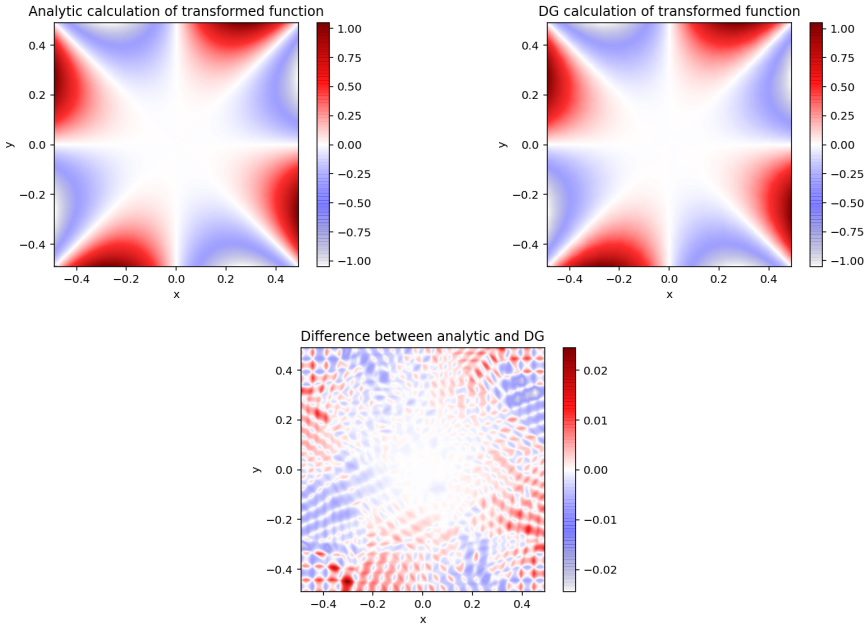


Fig. 4.1: Slices in the xy-plane ($z = 0$) of $(\boldsymbol{A}\cdot\nabla + \nabla\cdot\boldsymbol{A})f$ for $\boldsymbol{A} = \hat{z}\times\boldsymbol{r}$ computed analytically and using the operators derived in Section 3.1. The discretization uses a 4th order local basis. The difference between the analytic and DG calculations illustrate that our implementation of the magnetic field operator is working as intended.

$$-\frac{\partial V}{\partial z}\left(i\pi\sin(\pi y) + A_y\cos(\pi y)\right)\prod_{\mu\neq y}\cos(\pi\mu). \tag{4.4}$$

We then perform the same tests that we did with the magnetic field operator, for which exemplary results can be seen in Figure 4.3. We again find that our implementation of the SOC operator works as intended.

4.2. Perturbative tests. To test our implementation perturbatively, we consider an unperturbed Hamiltonian taking the form of a 3D particle in a box of unit length centered on the origin. In units with $\hbar = m_{\mathit{eff}} = 1$, the unperturbed energies and wave functions are simply

$$E_{\boldsymbol{n}} = \frac{\pi^2}{2}\sum_{\mu\in C} n_\mu^2 \quad\text{and}\quad \psi_{\boldsymbol{n}}(\boldsymbol{x}) = \sqrt{8}\prod_{\mu\in C}\sin\left(n_\mu\pi\left(\mu+\frac{1}{2}\right)\right). \tag{4.5}$$

First, we test the magnetic field operator. We consider a magnetic field with a simple vector potential of $\boldsymbol{A} = A_o\hat{z}\times\left(\boldsymbol{r} + \frac{\hat{x}+\hat{y}+\hat{z}}{2}\right)$ applied perturbatively to the non-degenerate ground state ($\boldsymbol{n} = \boldsymbol{1} = (1,1,1)$). The first order correction to this state is given as

$$\psi_{\boldsymbol{1}}^{(1)} = -i\sum_{\boldsymbol{n}\neq\boldsymbol{1}}\frac{\psi_{\boldsymbol{n}}}{E_{\boldsymbol{1}} - E_{\boldsymbol{n}}}\int_\Omega \mathrm{d}^3\mathrm{x}'\psi_{\boldsymbol{n}}\boldsymbol{A}\cdot\nabla\psi_{\boldsymbol{1}}. \tag{4.6}$$

For $\boldsymbol{A} = A_o\hat{z}\times\boldsymbol{r}$, the dominant corrections come from the $\boldsymbol{n} = (2,1,1)$ and $(1,2,1)$ terms, which is evident in Figure 4.4.

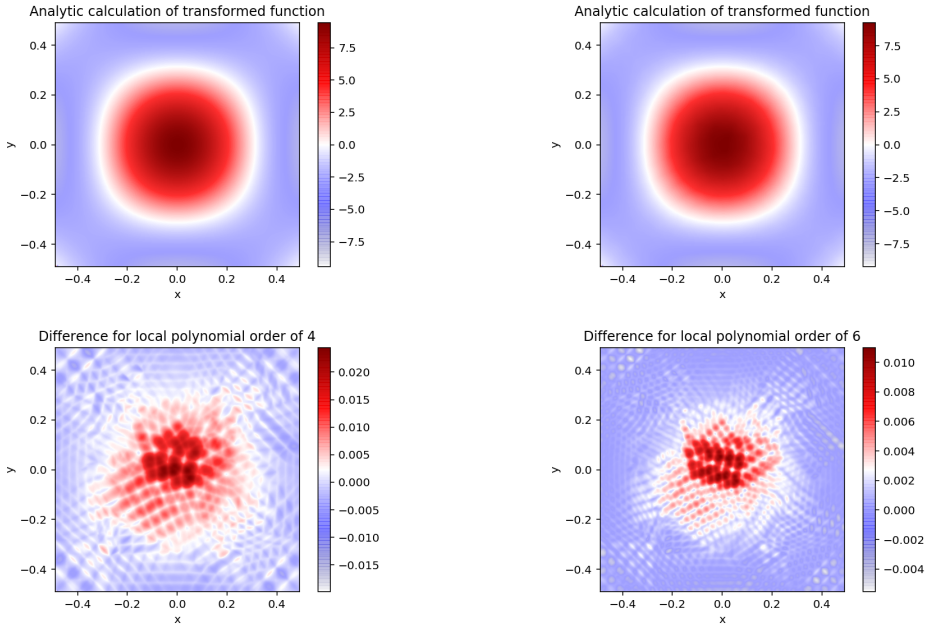


Fig. 4.2: Slices in the xy-plane ($z = 0$) of $(\boldsymbol{A} \cdot \nabla + \nabla \cdot \boldsymbol{A}) f$ for $\boldsymbol{A} = \sum_{\mu \in C} \hat{\mu} \sin(\pi \mu)$ computed analytically and using the operators derived in Section 3.1. The difference between the two calculations is shown for 4th (left) and 6th (right) order local basis sets, illustrating that the error decreases with an increase in the order of the local basis. Note the difference in the ranges for the two error plots.

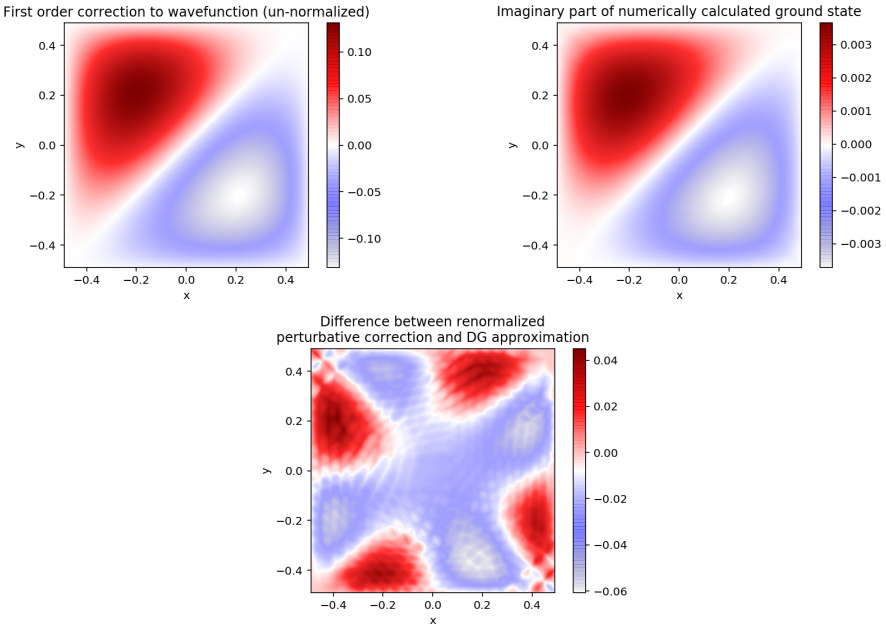


Fig. 4.4: Slices in the xy-plane ($z = 0$) of the correction to the wave function as calculated perturbatively (un-normalized) and as calculated by the DG method for $A_o = \frac{1}{100}$. The difference is calculated after renormalizing the two to have the same maximum value of 1. The error at any point in space is less than 6%. The perturbative expansion includes the 15 largest contributions to the correction.

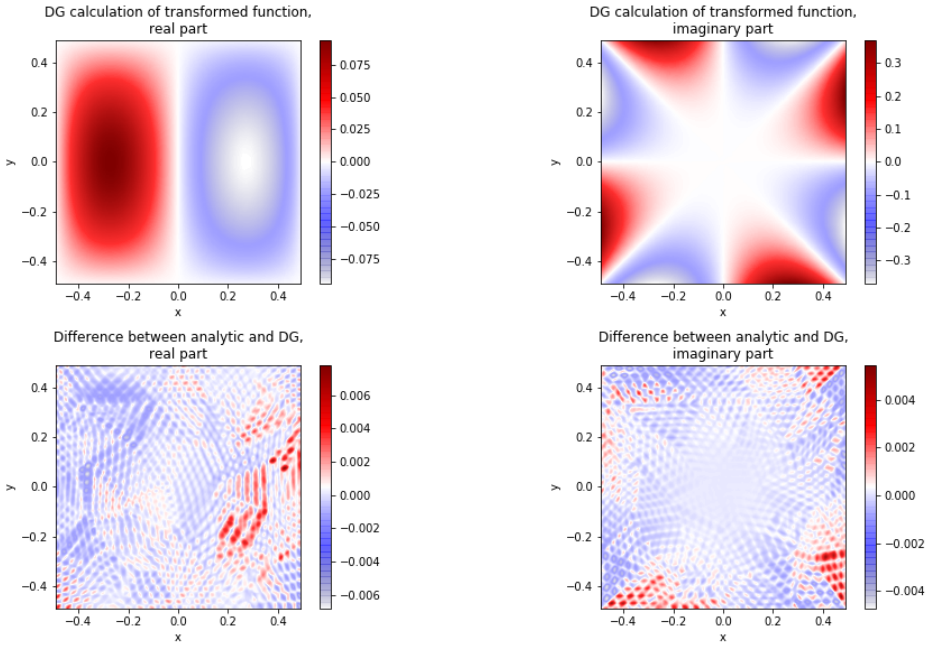


Fig. 4.3: Slices in the xy-plane ($z = 1/4$) of $(\nabla V \times \hat{\boldsymbol{p}})_x f$ for $\boldsymbol{A} = \hat{z} \times \boldsymbol{r}$ and $V = \frac{1}{2}\sum_{\mu \in C} \mu^2$ computed using the operators derived in Section 3.2. The difference between the analytically calculated and numerically calculated values are illustrated for 6th order local basis sets.

We test out the spatial part of the spin-orbit operator in exactly the same way, though excluding the spin of the particle forces us to focus on individual components of the vector operator $\nabla V \times \hat{\boldsymbol{p}}$. While applying the operator $(\nabla V \times \hat{\boldsymbol{p}})_x$ as a perturbation to the same particle in a box Hamiltonian may not seem to make sense physically if we don't have a constant V, the perturbation is still perfectly well-defined mathematically. The first order correction to the ground state is

$$\psi_{\mathbf{1}}^{(1)} = \sum_{\boldsymbol{n} \neq \mathbf{1}} \frac{\psi_{\boldsymbol{n}}}{E_{\mathbf{1}} - E_{\boldsymbol{n}}} \int_{\Omega} \mathrm{d}^3\mathrm{x}' \psi_{\boldsymbol{n}} \left(\nabla V \times \hat{\boldsymbol{p}}\right)_x \psi_{\mathbf{1}}. \tag{4.7}$$

For $V = \frac{V_o}{2}\sum_{\mu \in C} \mu^2$ and $\boldsymbol{A} = \boldsymbol{r}$, the first order contribution is again entirely imaginary as $(\nabla V \times \hat{\boldsymbol{p}})_x = -iV_o\left(y\frac{\partial}{\partial z} - z\frac{\partial}{\partial y}\right)$. The dominant contributions to first order correction are from the $\boldsymbol{n} = (1, 2, 4)$ and $\boldsymbol{n} = (1, 4, 2)$ terms, which is evident in Figure 4.5.

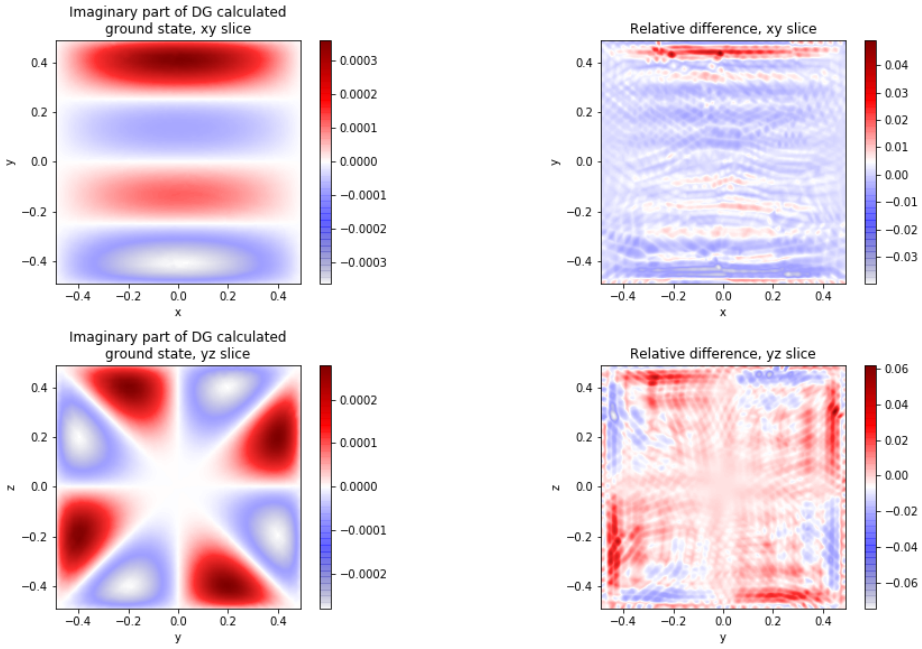


Fig. 4.5: Slices in the xy- ($z = 1/4$) and yz- ($x = 1/4$) planes of the correction to the wavefunction as calculated by the DG method for $V_o = \frac{1}{20}$. The difference is calculated after renormalizing the two to have the same maximum value of 1. The error at any point in space is less than 6%.

4.3. Analytic test. Finally, we verify our implementation for the solution of a 2D harmonic oscillator in a magnetic field, which serves as a toy model of a lithographic QD like the ones we intend to model using this capability. Without including SOC this problem has an analytic solution. While Laconic is centered around calculations for 3D systems, we can easily enough decouple the z-component of the Hamiltonian from the xy-component if we have the harmonic potential only in the xy-plane and a static magnetic field in the z-direction. For our example, we will use $\boldsymbol{A} = \frac{B}{2}\hat{z} \times \boldsymbol{r}$. The resulting eigenfunctions are separable in xy and z, with the xy solutions being of the form

$$\phi_{n,\ell}(r,\theta) = \frac{e^{i\ell\theta}}{a\sqrt{2\pi}} \sqrt{\frac{n!}{2^{|\ell|}(n+|\ell|)!}} \left(\frac{r}{a}\right)^{|\ell|} e^{-r^2/4a^2} L_n^{|\ell|}\left(\frac{r^2}{2a^2}\right) \tag{4.8}$$

where $a = \sqrt{\hbar/2m\sqrt{\omega^2 + e^2B^2/4m^2}}$ and L is a generalized Laguerre polynomial. To avoid any issues that a specific choice of gauge might bring to the phases of our states, we can simply compare $|\phi_{n,\ell}|^2$ for the analytic and DG solutions. Examples can be seen in Figure 4.6.

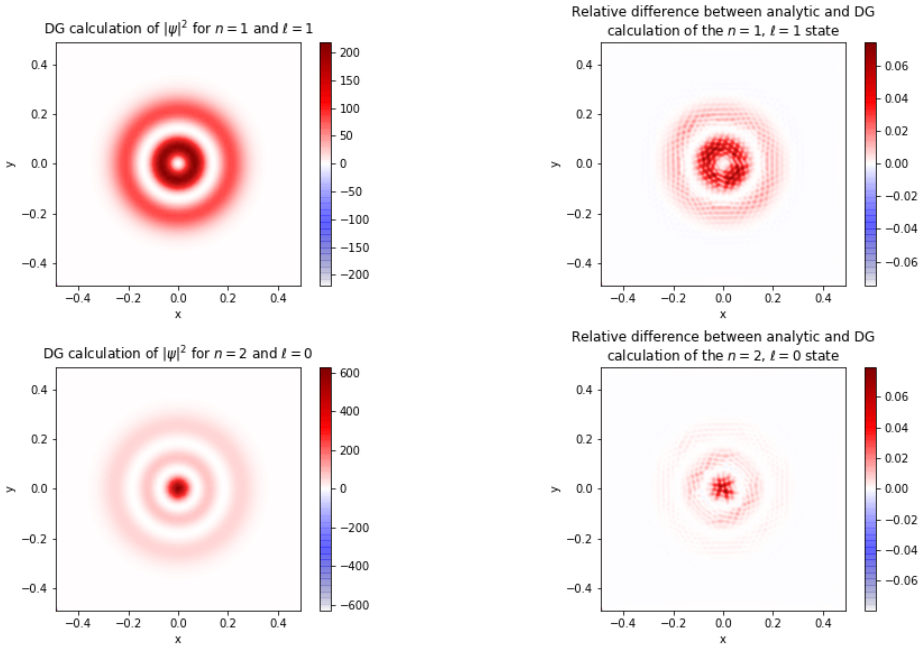


Fig. 4.6: Slices in the xy-plane ($z = 0$) of DG solutions for the $n = 1$, $\ell = 1$ (top) and $n = 2$, $\ell = 0$ (bottom) eigenfunctions. The relative errors with respect to the analytic solution are also illustrated, indicating that our implementation correctly predicts the eigenstates of a model of a QD in the presence of a magnetic field.

5. Summary and Future Work. In this report, we have developed an IPDG formulation of the Schrödinger-like equations governing effective mass theory that includes arbitrary magnetic fields and SOC. This formulation has been implemented in the Laconic software package and we have described the tests used to verify their correct operation. Future numerical work will involve a more rigorous error analysis and the implementation of basis function enrichment.

We have illustrated that our implementation reproduces an analytic calculation intended to model a lithographic QD in a magnetic field, suggesting that this capability is ready to be used for more realistic applications. We will be modeling upcoming experiments in Ge/SiGe QDs with the aim of predicting Rabi oscillation frequencies for electric-dipole spin resonance control of single hole spins. The ability to accurately predict this directly from device designs will allow us to determine which gate is best to use to control a given QD and also to optimize the gate layout for future devices. The work presented in this paper details an important step toward that goal.

REFERENCES

[1] J. K. Gamble, P. Harvey-Collard, N. T. Jacobson, A. D. Baczewski, E. Nielsen, L. Maurer, I. Montaño, M. Rudolph, M. Carroll, C. Yang, et al., *Valley splitting of single-electron si mos quantum dots*, Applied Physics Letters, 109 (2016), p. 253101.

[2] J. K. Gamble, N. T. Jacobson, E. Nielsen, A. D. Baczewski, J. E. Moussa, I. Montaño, and R. P. Muller, *Multivalley effective mass theory simulation of donors in silicon*, Physical Review B, 91 (2015), p. 235318.

[3] W. J. Hardy, C. Harris, Y.-H. Su, Y. Chuang, J. Moussa, L. Maurer, J.-Y. Li, T.-M. Lu, and D. R. Luhman, *Single and double hole quantum dots in strained ge/sige quantum wells*, arXiv preprint arXiv:1808.07077, (2018).

[4] P. Harvey-Collard, *P. harvey-collard, nt jacobson, m. rudolph, j. dominguez, ga ten eyck, jr wendt, t. pluym, jk gamble, mp lilly, m. pioro-ladrière, and ms carroll, nat. commun. 8, 1029 (2017).*, Nat. Commun., 8 (2017), p. 1029.

[5] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, Springer Science & Business Media, 2007.

[6] R. M. Jock, N. T. Jacobson, P. Harvey-Collard, A. M. Mounce, V. Srinivasa, D. R. Ward, J. Anderson, R. Manginell, J. R. Wendt, M. Rudolph, et al., *A silicon metal-oxide-semiconductor electron spin-orbit qubit*, Nature communications, 9 (2018), p. 1768.

[7] E. Kawakami, P. Scarlino, D. R. Ward, F. Braakman, D. Savage, M. Lagally, M. Friesen, S. N. Coppersmith, M. A. Eriksson, and L. Vandersypen, *Electrical control of a long-lived spin qubit in a si/sige quantum dot*, Nature nanotechnology, 9 (2014), p. 666.

[8] L. Lin, J. Lu, L. Ying, and E. Weinan, *Adaptive local basis set for kohn–sham density functional theory in a discontinuous galerkin framework i: Total energy calculation*, Journal of Computational Physics, 231 (2012), pp. 2140–2154.

[9] J. R. Petta, A. C. Johnson, J. M. Taylor, E. A. Laird, A. Yacoby, M. D. Lukin, C. M. Marcus, M. P. Hanson, and A. C. Gossard, *Coherent manipulation of coupled electron spins in semiconductor quantum dots*, Science, 309 (2005), pp. 2180–2184.

[10] J. J. Pla, K. Y. Tan, J. P. Dehollain, W. H. Lim, J. J. Morton, F. A. Zwanenburg, D. N. Jamieson, A. S. Dzurak, and A. Morello, *High-fidelity readout and control of a nuclear spin qubit in silicon*, Nature, 496 (2013), p. 334.

[11] L. Terrazos, A. Saraiva, X. Hu, M. Friesen, S. Coppersmith, B. Koiller, and R. B. Capaz, *Light-mass hole-spin qubits formed in a ge quantum well*, arXiv preprint arXiv:1803.10320, (2018).

[12] M. Veldhorst, J. Hwang, C. Yang, A. Leenstra, B. de Ronde, J. Dehollain, J. Muhonen, F. Hudson, K. M. Itoh, A. Morello, et al., *An addressable quantum dot qubit with fault-tolerant control-fidelity*, Nature nanotechnology, 9 (2014), p. 981.

AN EXCHANGE-CORRELATION FUNCTIONAL FOR BULK, SURFACE, AND CONFINEMENT PHYSICS

FRANCISCA SAGREDO * AND ATTILA CANGI†

Abstract. Density functional theory is cited over 30,000 times per year and has become one of the standard methods in electronic structure calculations due to its low cost, and ease of use. While there are many new approximations to the exchange-correlation functional appearing frequently in the literature, the need for an approximation which correctly captures the correct bulk, surface, and confinement physics is missing. Such physics is commonly found in d- and f- electron systems. These systems are common and relevant to DOE national labs, and are of the national interest. Below we develop a new exchange-correlation functional that captures this correct physics and demonstrate its accuracy on the jellium surface.

1. Introduction. The use of approximations in density functional theory (DFT) has become a standard tool in electronic structure theory, due to its accessibility, and low cost of calculations. This is especially true when compared to other standard methods such as configuration interaction (CI) [14], coupled cluster (CC) [6], GW [5], quantum Monte Carlo (QMC) [11]. While numerous approximations to the exchange-correlation (XC) functional are constantly appearing in the literature, many functional approximations still fail to capture the correct confinement physics. More specifically, XC functional approximations are known to fail for the "heavier" elements (e.g. transition metals) of the periodic table.

Thus the need of a functional approximation that fulfills these requirements becomes apparent. DFT approximations are used readily in material and molecular calculations since they provide structural information, vibrational properties, and even magnetic and electron polarizations; all things needed when investigating material systems. Due to the interest of the DOE's heavy element chemistry research [1], the need to accurately capture these material properties has been notably missing. Due to the inaccuracy in this regime, different approaches to remedy it have been investigated, including DFT + U [2,3], hybrid functionals [7,17,19], and dynamical mean field theory [12]. Below we propose an approximation to provide a solution to this problem. In particular we construct an XC functional to accurately capture bulk, solid, and confinement (BSC) physics of these elements. We focus on the confinement aspect which has been lacking in past approximations. We construct the BSC functional, run preliminary calculations using jellium surface reference data, and implement the functional into a quantum chemistry code.

2. The many body problem. Here we express the many-body problem within the realm of non-relativistic time-independent quantum mechanics, and use atomic units throughout with $\hbar = e = m_e = 1$. The total molecular Hamiltonian can be written as

$$\hat{H}_{mol} = \hat{T}_e(\mathbf{r}) + \hat{V}_{ee}(\mathbf{r}) + V_{en}(\mathbf{r}, \mathbf{R}) + \hat{V}_{nn}(\mathbf{R}) + \hat{T}_n(\mathbf{R}), \tag{2.1}$$

where $\hat{T}_e$ and $\hat{T}_n$ correspond to the kinetic energies of the electron and nuclear coordinates, respectively written in the traditional manner. Like wise $\hat{V}_{ee}$, $\hat{V}_{ne}$, and $\hat{V}_{nn}$ correspond to the electron-electron, nuclear-electron, and nuclear-nuclear potential operators written in the usual sense. Finally we note that $\mathbf{r}$ and $\mathbf{R}$ correspond to the electronic and nuclear coordinates, respectively.

Due to the complexity of the coupled terms, the molecular Hamiltonian quickly becomes difficult to solve, which is why the Born-Oppenheimer (BO) approximation is typically

*University of California, Irvine, fsagredo@uci.edu
†Sandia National Laboratories, acangi@sandia.gov

used [8,9]. Within the BO approximation, the nuclear and the electronic contributions of the Hamiltonian are separated, and are rewritten in a manner so that

$$\hat{H}_{mol} = \hat{H}_{BO}(\mathbf{r}, \mathbf{R}) + \hat{T}_n(\mathbf{R}). \tag{2.2}$$

Here, the $\hat{H}_{BO}$ corresponds to the BO Hamiltonian. The usefulness of the BO approximation arises from the large mass difference between electrons and the nucleus (electrons are ≈ 1800 times lighter). This large mass difference allows the electronic Hamiltonian to be solved for, independently. This is what is calculated in electronic structure theory, hence its name. Typically, $\hat{V}_{nn}(\mathbf{R})$ is first left out of the electronic Schrödinger equation since $\mathbf{R}$ acts as a parameter, and therefore $\hat{V}_{nn}(\mathbf{R})$ is a constant which shifts the eigenvalues by a constant amount. The solution of the electronic Hamiltonian then give rise to the corresponding wavefunctions, and potential energy surfaces for each electronic state.

The main problem in electronic structure methods arise from the electron-electron interaction in the electronic Hamiltonian. This term describes the repulsion between electrons within atoms and molecules. This term, and the configuration space of the wavefunction increase exponentially with the number of electrons, N. To explicitly demonstrate this problem you can imagine the many body electronic Hamiltonian for a single oxygen atom. Since the oxygen atom has 8 electrons, its wavefunction should have the form

$$\Psi_O(\mathbf{r}_1, ..., \mathbf{r}_8). \tag{2.3}$$

Thus meaning the wavefunction in 3 dimensions has 24 coordinates that must be evaluated. Clearly, this is already a difficult problem to solve, even though it is a trivial system that is not of interest. If this wavefunction were written for a material, or larger molecular system (e.g. a protein), its wavefunction becomes too complex to be solved computationally. This electron-electron interaction is approximated by the exchange-correlation (XC) functional in DFT.

3. Background on DFT. Density functional theory (DFT) is currently used across disciplines from chemistry and physics to materials science [10]. DFT is applied in a wide range of interdisciplinary fields from molecules and materials to even simulate inertial confinement fusion [10,24]. In its original formalism, DFT provides in principle an exact way to determine the ground-state energy of a given system, which becomes extremely valuable when studying chemical and physical systems. Starting with the Schrödinger equation

$$\{\hat{T} + \hat{V_{ee}} + \hat{V}\}\Psi = E\Psi, \tag{3.1}$$

the Hamiltonian is broken up in the typical manner where $\hat{T}$ is the kinetic energy, $\hat{V_{ee}}$ is the electron-electron repulsion operator, and $\hat{V}$ is the external potential, or the interaction between the nucleus and the electrons. Then by use of the variational principle the ground state energy can be found by the minimization over all normalized, antisymmetric trial wavefunctions,

$$E_{gs} = \min_{\Psi} \langle \Psi | \hat{H} | \Psi \rangle. \tag{3.2}$$

In the original Hohenberg-Kohn (HK) paper [13], (i) a one-to-one correspondence between the density $n(\mathbf{r})$, and the potential $v(\mathbf{r})$, proved that the ground-state properties of a given system can be uniquely determined by its density (for example, the ground state energy), and (ii) that a universal functional $F[n]$ exists. Although the HK theorem provided the foundation for the beginning of DFT it does not include the cases of degenerate ground states or non v representable densities; that is, ground state densities that cannot be defined

in terms of their potential. The generalized constrain search proposed by Levy [18], provides a clear and straightforward way in which to rewrite the variational principle to include the degeneracies and non v representability that the original HK theory did not include. Using the Levy formalism it is simple to write the energy minimization over all wavefunctions that yield the density

$$E = \min_{\Psi} \langle \Psi | \hat{T} + \hat{V_{ee}} + \hat{V} | \Psi \rangle = \min_{n} \{ F[n] + \int d\mathbf{r} v(\mathbf{r}) n(\mathbf{r}) \}, \tag{3.3}$$

given the universal functional $F[n]$ defined as

$$F[n] = \min_{\Psi \to n} \langle \Psi | \hat{T} + \hat{V_{ee}} | \Psi \rangle. \tag{3.4}$$

In practice we rarely solve this minimization over wavefunctions (except for very simple cases). When minimized over all densities $n(\mathbf{r})$, this immediately gives the ground state energy. In practice approximations must be used but it is important to remember that the theory is exact.

Kohn and Sham [16] provided the ground work for what is currently used and known in all modern DFT calculations as the Kohn-Sham (KS) scheme. In the KS scheme an auxiliary system of non-interacting electrons is defined which satisfies the non-interacting Schrödinger equation,

$$\{ -\frac{1}{2} \nabla^2 + v_s(\mathbf{r}) \} \phi_j(\mathbf{r}) = \epsilon_j \phi_j(\mathbf{r}), \tag{3.5}$$

with

$$n(\mathbf{r}) = \sum_{j=1}^{N} |\phi_j(\mathbf{r})|^2. \tag{3.6}$$

Here $v_s(\mathbf{r})$ is the KS potential, and the $\phi_j(\mathbf{r})$ correspond to the non-interacting orbitals, which are referred to as the KS orbitals. Similar to Hartree-Fock, single Slater determinants made up of KS orbitals are used to define the wavefunction of the system. The KS potential can also be written as

$$v_s(\mathbf{r}) = v(\mathbf{r}) + v_H[n](\mathbf{r}) + v_{XC}[n](\mathbf{r}), \tag{3.7}$$

where $\hat{V} = \sum_{i=1}^{N} v(\mathbf{r_i})$ is the external potential, $v_H(\mathbf{r})$ is the Hartree potential, and $v_{XC}(\mathbf{r})$ is the exchange-correlation potential. It should be noted that the Born-Oppenheimer approximation is assumed and atomic units are used throughout, where $\hbar = e = m_e = 1$. Following the KS formalism, it can be seen that the energy components of the KS scheme are simply a rearrangement of the traditional Schrödinger equation

$$E_{gs} = \min_{n} \{ T_s[n] + U[n] + V[n] + E_{XC}[n] \}, \tag{3.8}$$

with

$$v_{XC}[n](\mathbf{r}) = \frac{\partial E_{XC}}{\partial n(\mathbf{r})}. \tag{3.9}$$

Here T_s is the non-interacting kinetic energy, and the Hartree energy U which is also known as the Coulombic interaction, and $E_{XC}[n]$ is the exchange-correlation energy, with n being the ground-state density of the corresponding system. In general, T, T_s, and U are positive quantities, and can be thought of as trying to drive your system (atom) apart, and V, E_X, and E_C are negative trying to keep your system together. The exchange-correlation energy $E_{XC}[n]$ is what is approximated in DFT calculations, and what we focus on in this report.

4. Exchange-correlation approximations and the subsystem functional scheme. Within DFT there are several ways to approach the construction of an XC functional. For instance, it is possible to approach it using "Jacob's ladder" where in principle by adding exact conditions, new and improved functionals can be derived [20]. The ladder begins with the simplest approximation at the bottom, with the "heaven" of chemical accuracy at the top (that will most likely not be reached). The bottom step of this ladder begins with the local density approximation (LDA) [16]

$$E_{XC}^{LDA}[n] = \int d\mathbf{r} n(\mathbf{r}) \epsilon_{XC}^{LDA}[n](\mathbf{r}), \tag{4.1}$$

with ϵ_{XC}^{LDA} defined as the exchange-correlation energy per particle of the uniform electron gas. Its exchange component is explicitly defined as

$$\epsilon_{X}^{LDA} = -\frac{3}{2\pi}(3\pi^2 n(\mathbf{r}))^{1/3}, \tag{4.2}$$

whereas its correlation component, ϵ_{C}^{LDA}, is a parametrization [21, 22, 25] of the correlation energy in the uniform electron gas [11]. LDA was derived using the uniform electron gas as the reference system, and naturally does well in approximating systems with uniform, slowly varying densities, like that of solids, phonon frequencies, and metal surface energies. Even though LDA is the most basic, and first functional approximation proposed in DFT, it is still used readily and provides the basis of many other functionals.

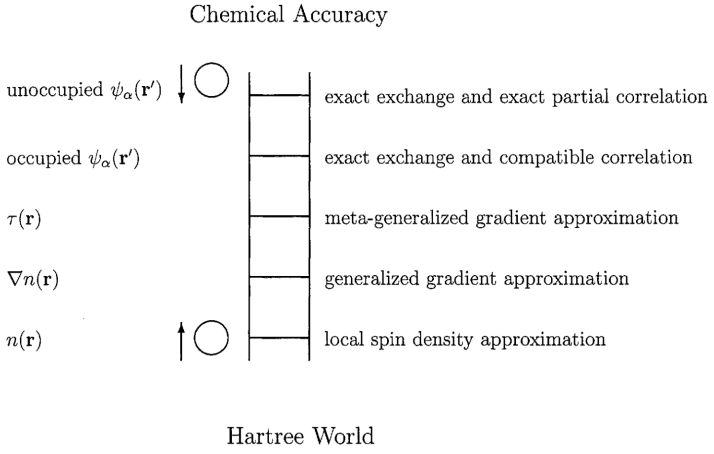


Fig. 4.1: Jacob's ladder of chemical accuracy. Figure replicated from [20]

After LDA the natural next "step" in XC functional approximations is the generalized gradient approximation (GGA). GGAs use the gradient of the density, and can be expressed as

$$E_{XC}^{GGA}[n] = \int d\mathbf{r} n(\mathbf{r}) \epsilon_{XC}^{GGA}[n, |\nabla n|](\mathbf{r}). \tag{4.3}$$

The ϵ_{XC}^{GGA} term also includes higher order terms. Although GGAs provide an improvement to LDA, it still has many limitations and does not accurately approximate bond length and ionization potentials.

Finally, meta GGAs (MGGA) are the next "step up " in Jacob's ladder, and include the kinetic energy density of the KS system, τ_s. The kinetic energy density τ_s is used since it allows the functional to use varying types of electron-electron interactions. MGGAs take the form

$$E_{XC}^{MGGA}[n] = \int d\mathbf{r} n(\mathbf{r}) \epsilon_{XC}^{MGGA}[n, |\nabla n|, \tau_s](\mathbf{r}). \tag{4.4}$$

MGGAs do provide a substantial correction to XC approximations. Here we begin with a specific MGGA, AM05 [4]. The exchange-correlation functional AM05, by Armiento and Mattsson provides an alternate route to improving XC functionals within DFT. AM05 uses the subsystem functional approach [15], which in essence uses multiple reference systems as a means to improve the functional. Subsystem functionals use an interpolation index to "pick out" the correct contributions depending on the system being studied. The use multiple reference systems is particularly important since many XC approximations have limited applicability, and only work for very specific systems, notably failing outside the specified conditions. The subsystem functional approach uses [15],

$$E_{XC} = \sum_j \int_{\Omega_j} d\mathbf{r}_j n(\mathbf{r}) \epsilon_{XC}^j[n](\mathbf{r}). \tag{4.5}$$

This is simply a sum over the subsystems j where Ω_j denotes its volume. In particular AM05 uses two reference systems, the jellium gas (LDA), and the Airy gas. The jellium gas has a constant potential, where as the Airy gas has a linear potential. Using these two reference systems the AM05 functional correctly captures the physics of bulk solids (jellium), and the correct surface physics (Airy gas). The interpolation index is also a vital part of its success, since it can "pick" out the correct contributions depending on the system being studied. The XC energy per particle of the reference systems can be easily expressed as

$$\epsilon_{XC}^{bulk}[n](\mathbf{r}) = \epsilon_{XC}^{LDA}[n](\mathbf{r}) \tag{4.6}$$

and

$$\epsilon_{XC}^{surf}[n](\mathbf{r}) = \epsilon_{XC}^{AM05}[n](\mathbf{r}) \ . \tag{4.7}$$

Although AM05 provides a good approximation to the XC functional for bulk, and surface physics, it still misses the confinement physics found in d- and f- electron systems. Therefore a third reference system must be used and incorporated in the construction of the new BSC functional, in order to systematically make an improved functional.

5. Harmonic oscillator gas reference system. As mentioned, a third reference system is needed to accurately capture the confinement physics in the BSC XC functional. Here we choose the harmonic oscillator gas (HOG), which has a harmonic potential only confined in the z- axis, while still being infinite in the other two dimensions. The harmonic effective potential is defined as

$$v_{eff}(z) = \frac{\omega^2 z^2}{2}, \tag{5.1}$$

with oscillations, ω. Using this effective potential, solving the non-interacting KS Schrödinger equation yields plane wave solutions for the KS orbitals. The analytic KS orbitals can be expressed as

$$\phi_j(\bar{z}) = \left(\frac{1}{2^j j! l\sqrt{\pi}}\right)^{1/2} H_j(\hat{z}) \exp\left(-\bar{z}^2/2\right) \tag{5.2}$$

with the expected Hermite polynomials, H_j, of the the j-th eigen state. l can be defined as $l = \sqrt{1/\omega}$, and $\bar{z} = \mathbf{r}/l$, with eigen values $\epsilon_j = l^{-2}(j + 1/2)$. The HOG denisity can be written as

$$[l^3 n(\bar{z})] = \pi^{-3/2} \sum_{j=0}^{N} \frac{1}{2^j j!} H_j^2(\bar{z})(\alpha - j) \exp\left(-z^2\right). \tag{5.3}$$

The α parameter is defined in terms of the chemical potential, $\mu = (\alpha+1/2)/l^2$. It determines the amount of confinement in the HOG and is dependent on the number of electrons $\alpha = N_e$ in the system.

Finally, within the HOG we write the new expression and functionals in terms of the electron localization function (ELF). ELF is a number that varies between $0 \leq ELF \leq 1$, and measures the probability of finding an electron surrounding a reference electron at a given point. It can easily be defined starting with

$$D_\sigma(\mathbf{r}) = \tau_\sigma - \frac{(\nabla n_\sigma(\mathbf{r}))^2}{n_\sigma(\mathbf{r})}, \tag{5.4}$$

with τ_σ as the kinetic energy density, with some spin density, n_σ. ELF also uses the uniform electron gas as a reference system

$$D_\sigma^0 = \frac{3}{5}(6\pi^2)^{2/3} n_\sigma^{5/3}(\mathbf{r}). \tag{5.5}$$

Using these two expressions it is possible to define a ratio

$$\chi_\sigma = \frac{D_\sigma(\mathbf{r})}{D_\sigma^0(\mathbf{r})}, \tag{5.6}$$

from which ELF can be defined

$$ELF(\mathbf{r}) = \frac{1}{1 + \chi_\sigma^2(\mathbf{r})}. \tag{5.7}$$

The HOG functional expressions are written in terms of this ELF parameter, such that the confinement physics can be captured correctly. When the value of $ELF = 1/2$, it is in a jellium state, when $ELF = 0$ the system behaves like a surface electron, and finally when $ELF = 1$ the electron is completely localized, and in the confinement regime. Like in the case of AM05, the XC energy per particle of the confinement regime can be expressed as

$$\epsilon_{XC}^{conf}[n](\mathbf{r}) = \epsilon_{XC}^{HOG}[n, |\nabla|n, ELF](\mathbf{r}). \tag{5.8}$$

6. Results: Jellium surface calculations. Here we used the jellium surface energy data from RPA calculations as the basis to test the new BSC XC functional [23]. The jellium surface is simply a slab of uniform electron gas, with some set thickness, d. It is useful to use this surface energy data as it provides a sufficient model for surface physics, whereas traditional jellium only captures the correct bulk contributions. All of the reference data calculated below, was determined with respect to the the Wigner-Seitz radius, $r_s = [3/(4\pi n)]^{1/3}$. To better understand how ELF works, a figure of ELF as a function of z is seen below. In particular we plot the ELF function with varying values of r_s. The confinement region is contained mostly on the range of $[-3, 0]$.

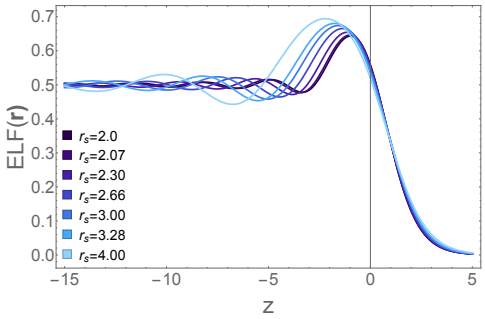


Fig. 6.1: Plot of ELF as a function of z for varying values of r_s

Two interpolations were determined and tested in the functional construction. These interpolation indices are,

$$\chi_{AC} = 1 - \frac{1}{(\eta_{2,\alpha} ELF)^{2\eta_{1,\alpha}} + 1}, \tag{6.1}$$

$$\chi_{FS} = \frac{\eta_{1,\beta}(ELF)^{16}}{1 + \exp(-\eta_{2,\beta}(ELF - 1)^2)}. \tag{6.2}$$

Here we define $\eta_{1,\alpha} = 1.781$ and $\eta_{2,\alpha} = 98.874$ as. Likewise $\eta_{1,\beta} = 10758$ and $\eta_{2,\beta} = -13.06$. These quantities were optimized using the jellium surface data on *Mathematica*. Now it is possible to construct the BSC functional. We begin with the exchange contribution,

$$\epsilon_X^{BSC} = \epsilon_X^{surf}(1 - \chi_i) + \epsilon_X^{conf}\chi_i, \tag{6.3}$$

with χ_i as the two different interpolations found. Likewise the full form of the functional can be expressed as:

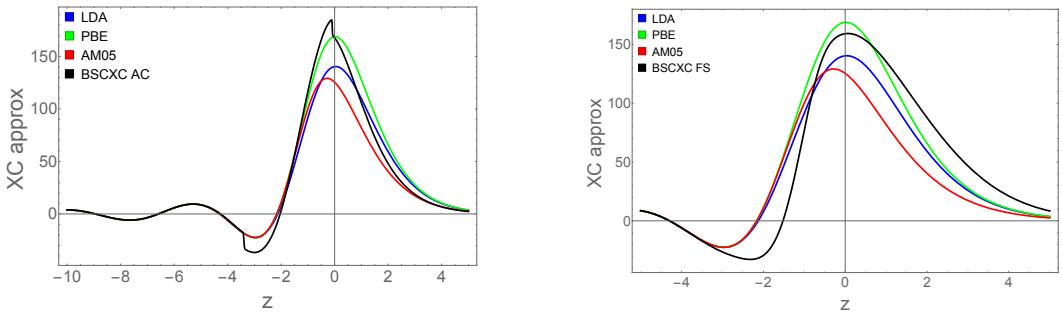
$$\epsilon_{XC}^{BSC}[n, |\nabla n|, ELF](\mathbf{r}) = \epsilon_{XC}^{BSC}[n, |\nabla n|, ELF](\mathbf{r}) + \epsilon_{XC}^{HOG}[n, |\nabla n|](\mathbf{r}). \tag{6.4}$$

The correlation component of the functional was approximatedby a scaled version of LDA correlation, similar to the construction in the AM05 functional.

Our tests indicate (Table 6.2) that the BSC functional captures the bulk, surface, and confinement physics of the jellium surface well. With a mean absolute error (MARE) of 0.024% and 0.038%, both versions of the BSC functional are significantly more accurate than the LDA and standard semilocal functionals (PBE and AM05). The XC surface energy densities corresponding to the two versions of the BSC functional are contrasted in Figures 6.3a and 6.3b.

r_s	σ_{xc}^{LDA}	σ_{xc}^{PBE}	σ_{xc}^{AM05}	$\sigma_{xc}^{BSC,AC}$	$\sigma_{xc}^{BSC,FS}$	EXACT
2.00	3353.58	3263.58	3402.82	3415.02	3412.84	3413
2.07	2960.12	2879.06	3004.37	3015.26	3014.61	3015
2.30	2018.52	1959.97	2050.54	2058.19	2059.64	2060
2.66	1187.35	1150.46	1208.04	1212.34	1213.99	1214
3.00	763.394	738.509	777.938	780.025	781.434	781
3.28	548.903	530.445	560.16	560.852	562.269	563
4.00	261.145	251.855	267.61	265.919	268.127	268
MARE	2.15%	5.17%	0.377%	0.024%	0.038%	

Fig. 6.2: Benchmarks of BSC functional compared to various other functional approximations relative to the surface jellium data with r_s corresponding to the Wigner-Seitz radius and MARE defined as the mean absolute relative error.



(a) BSC functional with the χ_{AC} interpolation.

(b) BSC functional with the χ_{FS} interpolation.

Fig. 6.3: Surface energy plots comparing various approximations.

7. Results: Implementation. After the benchmarks on the jellium surface, the BSC functional was implemented into the ELK code. ELK is a quantum mechanical code which is used here to benchmark the BSC functional on materials. Due to the nature of the code implementation, we do not describe specific procedures here.

8. Conclusions. We have explained the need of an accurate XC functional that correctly captures, bulk, solid, and confinement physics. Its need is vital for chemical accuracy of the "heavier" elements of the periodic table with d- and f- orbitals. The jellium surface benchmarks are very promising, with MARE of $\approx 0.024\%$. Current and future work includes implementing the code into the widely used *libxc* library. This will allow the BSC functional to be tested extensively on materials of interest, and also be compared with new functional approximations.

REFERENCES

[1] D. A. Andersson, G. Baldinozzi, L. Desgranges, D. R. Conradson, and S. D. Conradson, *Density functional theory calculations of uo2 oxidation: Evolution of uo2+x, u4o9–y, u3o7, and u3o8*, Inorganic Chemistry, 52 (2013), pp. 2769–2778. PMID: 23406007.

[2] V. I. Anisimov, I. V. Solovyev, M. A. Korotin, M. T. Czyżyk, and G. A. Sawatzky, *Density-functional theory and nio photoemission spectra*, Phys. Rev. B, 48 (1993), pp. 16929–16934.

[3] V. I. Anisimov, J. Zaanen, and O. K. Andersen, *Band theory and mott insulators: Hubbard u instead of stoner i*, Phys. Rev. B, 44 (1991), pp. 943–954.

[4] R. Armiento and A. E. Mattsson, *Functional designed to include surface effects in self-consistent density functional theory*, Phys. Rev. B, 72 (2005), p. 085108.

[5] F. Aryasetiawan and S. Biermann, *Generalized hedin's equations for quantum many-body systems with spin-dependent interactions*, Phys. Rev. Lett., 100 (2008), p. 116402.

[6] R. J. Bartlett and M. Musiał, *Coupled-cluster theory in quantum chemistry*, Rev. Mod. Phys., 79 (2007), pp. 291–352.

[7] A. D. Becke, *Density-functional exchange-energy approximation with correct asymptotic behavior*, Phys. Rev. A, 38 (1988), pp. 3098–3100.

[8] M. Born and K. Huang, *Dynamical theory of crystal lattices*, American Journal of Physics, 23 (1955), pp. 474–474.

[9] M. Born and R. Oppenheimer, *Zur quantentheorie der molekeln*, Annalen der Physik, 389, pp. 457–484.

[10] K. Burke, *Perspective on density functional theory*, J. Chem. Phys., 136 (2012).

[11] D. M. Ceperley and B. J. Alder, *Ground state of the electron gas by a stochastic method*, Phys. Rev. Lett., 45 (1980), pp. 566–569.

[12] A. Georges, G. Kotliar, W. Krauth, and M. J. Rozenberg, *Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions*, Rev. Mod. Phys., 68 (1996), pp. 13–125.

[13] P. Hohenberg and W. Kohn, *Inhomogeneous electron gas*, Phys. Rev., 136 (1964), pp. B864–B871.

[14] P. J. Knowles and H.-J. Werner, *Internally contracted multiconfiguration-reference configuration interaction calculations for excited states*, Theoretica chimica acta, 84 (1992), pp. 95–103.

[15] W. Kohn and A. E. Mattsson, *Edge electron gas*, Phys. Rev. Lett., 81 (1998), pp. 3487–3490.

[16] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev., 140 (1965), pp. A1133–A1138.

[17] C. Lee, W. Yang, and R. G. Parr, *Development of the colle-salvetti correlation-energy formula into a functional of the electron density*, Phys. Rev. B, 37 (1988), pp. 785–789.

[18] M. Levy, *Universal variational functionals of electron densities, first-order density matrices, and natural spin-orbitals and solution of the v-representability problem*, Proceedings of the National Academy of Sciences of the United States of America, 76 (1979), pp. 6062–6065.

[19] J. P. Perdew, K. Burke, and M. Ernzerhof, *Generalized gradient approximation made simple*, Phys. Rev. Lett., 77 (1996), pp. 3865–3868.

[20] J. P. Perdew and K. Schmidt, *Jacob's ladder of density functional approximations for the exchange-correlation energy*, AIP Conference Proceedings, 577 (2001), pp. 1–20.

[21] J. P. Perdew and Y. Wang, *Accurate and simple analytic representation of the electron-gas correlation energy*, Phys. Rev. B, 45 (1992), pp. 13244–13249.

[22] J. P. Perdew and A. Zunger, *Self-interaction correction to density-functional approximations for many-electron systems*, Phys. Rev. B, 23 (1981), pp. 5048–5079.

[23] J. M. Pitarke and A. G. Eguiluz, *Jellium surface energy beyond the local-density approximation: Self-consistent-field calculations*, Phys. Rev. B, 63 (2001), p. 045116.

[24] J. C. Smith, F. Sagredo, and K. Burke, *Warming Up Density Functional Theory*, Springer Singapore, Singapore, 2018, pp. 249–271.

[25] S. H. Vosko, L. Wilk, and M. Nusair, *Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis*, Can. J. Phys., 58 (1980), pp. 1200–1211.