Parallel ODETLAP for Terrain Compression and Reconstruction

Jared Stookey
ECSE Dept, RPI
jstookey@jstookey.com

Zhongyi Xie
CS Dept, RPI
xiez@rpi.edu

Barbara Cutler
CS Dept, RPI
cutler@cs.rpi.edu

W. Randolph Franklin
ECSE Dept, RPI
frankwr@rpi.edu

Dan Tracy
CS Dept, RPI
tracyd@rpi.edu

Marcus V. A. Andrade
DPI - Univ. Fed. Vicosa -
Brazil *and* RPI
marcus.ufv@gmail.com

ABSTRACT

We introduce a parallel approximation of an Over-determined Laplacian Partial Differential Equation solver (ODETLAP) applied to the compression and restoration of terrain data used for Geographical Information Systems (GIS). ODET-LAP can be used to reconstruct a compressed elevation map, or to generate a dense regular grid from airborne Light Detection and Ranging (LIDAR) point cloud data. With previous methods, the time to execute ODETLAP does not scale well with the size of the input elevation map, resulting in running times that are prohibitively long for large data sets. Our algorithm divides the data set into patches, runs ODET-LAP on each patch, and then merges the patches together. This method gives two distinct speed improvements. First, we provide scalability by reducing the complexity such that the execution time grows almost linearly with the size of the input, even when run on a single processor. Second, we are able to calculate ODETLAP on the patches concurrently in a parallel or distributed environment. Our new patch-based implementation takes 2 seconds to run ODETLAP on an 800×800 elevation map using 128 processors, while the original version of ODETLAP takes nearly 10 minutes on a single processor (271 times longer). We demonstrate the effectiveness of the new algorithm by running it on data sets as large as 16000×16000 on a cluster of computers. We also discuss our preliminary results from running on an IBM Blue Gene/L system with 32,768 processors.
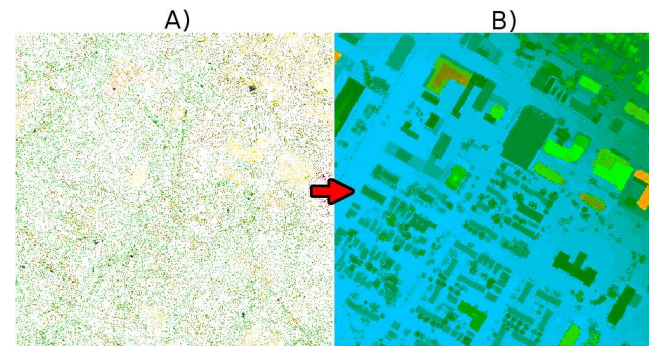
Categories and Subject Descriptors

I.3.5 [**Computing Methodologies**]: Computer Graphics—*Computational Geometry and Object Modeling*

Keywords

GIS, LIDAR, PDE solver, parallel computation, terrain modeling, terrain elevation data set compression, terrain interpolation

Figure 1: Our parallel version of ODETLAP was used to generate a dense 10000×10000 raster (B) from raw urban LIDAR data consisting of 13 million points (A) quickly on a cluster of 128 2.6 GHz AMD Opteron processors.

1. INTRODUCTION

Due to recent advances in GIS data acquisition methods such as LIDAR and satellite imagery, vast amounts of GIS data is being collected at such a fast rate that storing and processing all of this data currently poses a problem, and will increasingly pose a problem in the near future. We can help to reduce these problems in several ways. First, we can reduce the size of the stored data by inventing new compression techniques that are optimized for GIS data. Second, we can process data more quickly using new approximation techniques that will reduce the time required for operating on larger data sets. Third, by taking advantage of machines such as multi-processor and multi-core machines, as well as computing clusters and supercomputers, we will be able to transform vast amounts of raw data collected in the field into strategic planning intelligence in a short amount of time. In this paper, we introduce a method to accelerate the process of ODETLAP terrain compression and reconstruction [13] that demonstrates each of these.

ODETLAP has proven to be an effective method for generating data that closely represents the original terrain by filling in the unknown points. This feature can be used to generate raster data from sparse point cloud data, or to reconstruct an elevation map from a small subset of points. Using ODETLAP as a black box, we have developed terrain compression algorithms to reduce the storage space for large terrains. However, ODETLAP suffers from a scalabil-
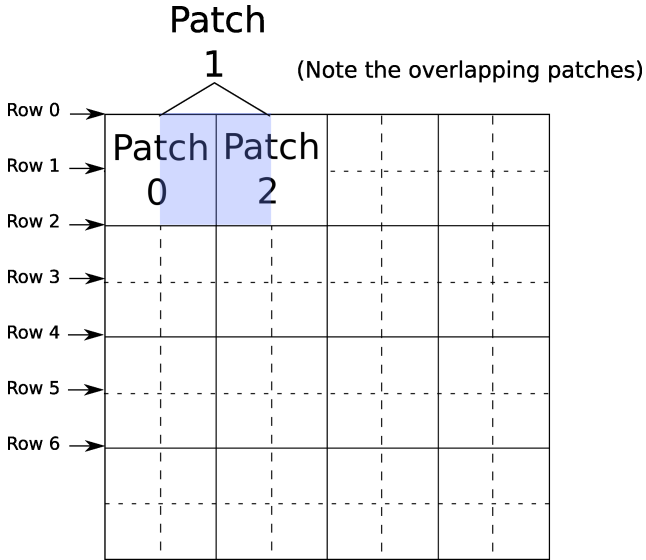
Figure 2: We apply ODETLAP to overlapping layers of patches. This example shows a 400x400 grid with 49 100x100 patches. In this example, 16 patches would be enough to cover the entire heightmap without any overlap, but we would see errors at the patch edges. Instead, we run ODETLAP on 49 overlapping patches.

Figure 3: Our final step is to merge the overlapping patches (A) into the complete reconstructed elevation map (B).

ity issue. As the dimensions of the elevation map increase, the numerical complexity of ODETLAP causes the computation time to increase quadratically with the number of pixels (Figure 5). In this paper, we describe an approximation for ODETLAP in which we divide the terrain into overlapping patches which ODETLAP can process quickly (Figure 2), and later merge the results together as shown in Figure 3. This method, which we call *patch-based ODETLAP*, quantizes the numerical complexity of ODETLAP, so that the execution time grows linearly with the size of the input matrix. Also, by dividing the input matrix into separate patches on which ODETLAP is run, we are able to take advantage of parallel and distributed systems to calculate multiple patches simultaneously. By combining both of these advantages, we are able to gain tremendous speed improvements for large terrain maps when compared to the original version, which we call *non-patch ODETLAP*.

Figure 1 provides a powerful demonstration of ODETLAP's ability to fill in unknown points. Given over 13 million sparse point cloud data points generated using airborne LIDAR, ODETLAP generates a very accurate representation of the original urban elevation map. Using our parallel patch-based ODETLAP, this calculation took 33 minutes and 22 seconds on 128 processors in a cluster of 2.6 GHz AMD Opterons. Extrapolating from the data in Figure 5, the same calculation would have taken more than 179 days using the non-patch version of ODETLAP on a single processor.

Parallel ODETLAP greatly improves the efficiency and still compares favorably with the non-patch ODETLAP in reconstruction accuracy. Because we provide generous overlap between the patches and ensure that sufficient known data samples are shared between the patches, the result of
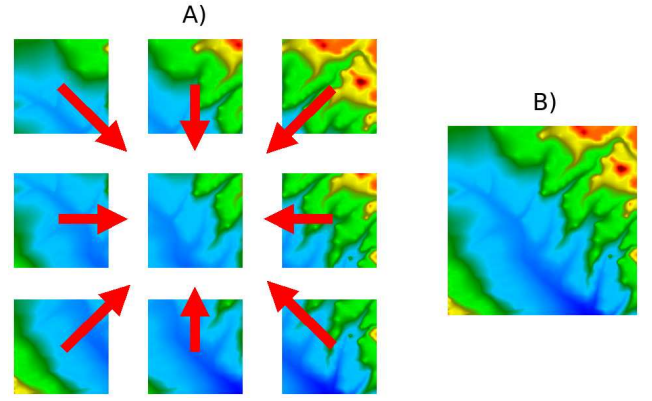
the patch-based parallel ODETLAP matches the non-patch version of ODETLAP within 0.1%.

Our contributions are as follows:

- A partitioning scheme for calculating ODETLAP that greatly reduces the overall numerical complexity for large elevation maps.

- A method for distributing and merging data partitions in order to calculate ODETLAP in a parallel or distributed environment.

- Demonstration of our algorithm for calculating ODETLAP on much larger terrains than what was previously possible with nearly identical results.

2. RELATED WORK

Research on parallel computing in GIS began to play an important role in the 1990s [2] as parallel computing technology became more widely available. Research has been done on parallelizing existing algorithms like line simplification by Mower [10], polygon line shading by Roche et al [11] and processing heterogeneous networks for GIS by Clematis [1]. Research at the Edinburgh Parallel Computing Centre (EPCC) emphasized creating new parallel libraries to support high performance GIS data models [2, 6, 9].

A newer approach to the problem of parallelizing GIS operations was taken in [7] in which the GRASS GIS application was modified to operate in a clustered environment. Ichikawa et al [12] demonstrate an iterative data partitioning scheme for parallelizing a PDE solver. Griebel et al [5] made excellent progress using parallel multigrid to solve PDE's. More recently, we have seen research in image processing that involves large linear systems on huge images using a streaming multigrid that can benefit by running on parallel systems [8] in such a way that could be adapted to problems in GIS.

We extend these important pieces of research to over-determined PDE's. We specifically target GIS applications and provide an in-depth analysis of the quality-cost trade-off associated with partitioning very large elevation maps.

In this paper, we propose a parallel terrain compression algorithm that is capable of processing terrain maps of size 16000×16000 and larger in a reasonable amount of time.

2.1 The ODETLAP Solver

The Over-determined Laplacian Approximation (ODETLAP), [13] is an extension to the Laplacian equation:

$$4z_{xy} = z_{x-1,y} + z_{x+1,y} + z_{x,y-1} + z_{x,y+1} \qquad (1)$$

This equation states that for every non-border point identified by coordinate (x, y) in the elevation matrix, the elevation z_{xy} is equal to the average of its neighbors. The handling of border points forms a special case and is omitted due to the lack of deep theoretical interest. This equation by itself is unable to represent local maxima in terrain modeling [4], thus we also include a second equation:

$$z_{xy} = h_{xy} \qquad (2)$$

Equations 1 and 2 form a basis for our over-determined linear system. The system's input is a set of points (x, y, z) which specify the elevation of certain locations (x, y). For those locations with known elevation, we have both equations, and for the rest of the locations, only equation 1 is specified. The relative importance of these two sets of equations is determined by a parameter R during the interpolation process. Weighting equation 2 over equation 1 results in a more accurate surface which sacrifices smoothness, while weighting equation 1 over equation 2 gives us a smoother surface that interpolates the known points.

We use ODETLAP when we have incomplete information about the actual elevation matrix. The known value and the Laplacian constraint (the average of its neighbors) can be used to interpolate the elevation value for every unknown and known point. In this way, ODETLAP can be considered as a solver whose input is a set of known points (x, y, z) and an interpolation parameter R and output is the Digital Elevation Model (DEM) matrix of the complete terrain. ODETLAP has several benefits that are ideal for terrain data, including the ability to handle continuous as well as broken contour lines of elevations, processing kidney-bean-shaped contours without giving fictitious results at regions inside, and the ability to infer local maxima from a series of contours.

2.2 ODETLAP-based Terrain Compression

Since ODETLAP is capable of reconstructing the whole DEM matrix from a few sparse input points (x, y, z), it can be used as a decompressor in the ODETLAP-based compression algorithm. In order to reduce the amount of space required to store the data, only a limited subset of the elevations from the original elevation data are stored. Later, ODETLAP is used to lossily reconstruct the elevation map by filling in the missing points.

Figure 4 presents the flow chart of the algorithm [4]. The DEM first undergoes a point selection which picks a subset of posts, S, as input to the ODETLAP solver. ODETLAP selects points that are deemed important to the accurate reconstruction of the terrain such as contour lines, border points, or any other available points. The points can consist of a sparse point cloud, a regular grid, or a mix between the two. The ODETLAP solver reconstructs from S the whole DEM matrix of elevations giving us an initial approximation
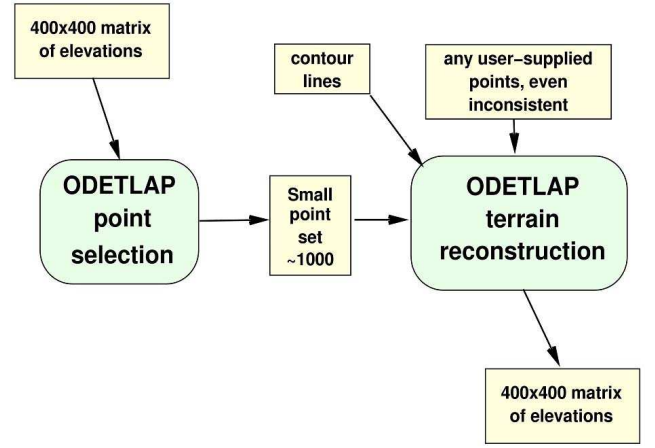
Figure 4: ODETLAP algorithm: Square boxes represent data and curved boxes (light green) represent operations.

	Mtn1	Mtn2	Mtn3
Elev range	1040m	953m	788m
St. Dev.	146.0	152.4	160.7
Orig size	320KB	320KB	320KB
Compr. size	9744B	9670B	9895B
Compr. ratio	33:1	33:1	32:1
# pts selected	4160	4160	4160
Elev. RMS error	9.48	9.55	9.68
Elev. RMS perc.	0.91%	1%	1.23%
Slope RMS error	8.34°	8.36°	7.87°

Table 1: Results from compressing three mountainous terrain data sets using non-patch ODETLAP.

of the elevation matrix. We initially pick a very small subset of points ($|S| \leq 1000$ in a 400×400 elevation matrix), yielding an approximation with insufficient accuracy. After obtaining the initial approximation, iterative refinement is performed to insert additional elevation values where the reconstructed surface most poorly matches the original data. The refinement steps end when the overall RMS elevation error is below the specified threshold. Table 1 summarizes the ODETLAP algorithm's compression performance on three mountainous terrain samples of size 400×400.

2.3 Regular and Irregular Point Data

In this paper we demonstrate ODETLAP in two different reconstruction scenarios. In Figure 1 a LIDAR point cloud with approximately 13 million points is inflated to a 100 million point dense grid. In the other examples in the paper we reconstruct the terrain from a compressed subset of points consisting of 1% of the total points, uniformly sampled in a regular grid. For example, a 400×400 grid consisting of 160000 points will be represented by 1600 values in its compressed form. With the ODETLAP compression scheme we can optionally augment the regular grid with additional important points to reduce the error in the reconstructed elevation map, however we do not concern ourselves with this in the context of this paper because we are focusing on the
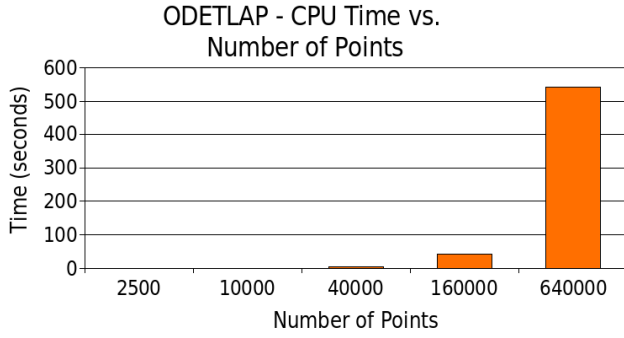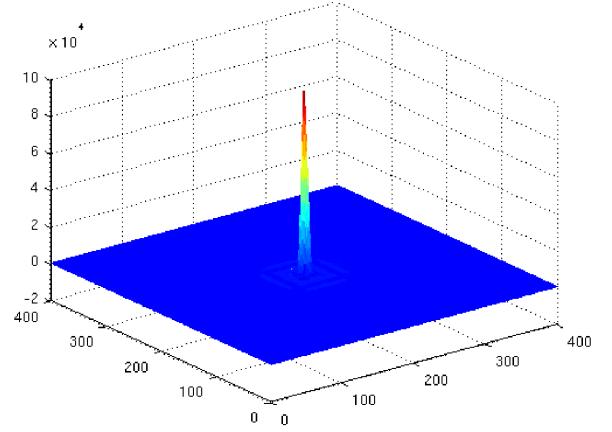
Figure 5: ODETLAP performance on a single large patch on grid with 1% known (original) points. The "Number of Points" refers to the number of points in the reconstructed grid. The calculation time grows quadratically with the number of points, and it takes nearly 10 minutes to process an 800×800 grid.

Figure 6: Altering a single known point in the terrain has a limited radius of impact during reconstruction. This plot shows the per-pixel difference between the ODETLAP solution (using a 1% subsampling of a 400×400 elevation map with elevations ranging from 219 to 1040) and the ODETLAP solution for the same data with the central point edited by setting its elevation to 100,000. The difference in the solutions was restricted to a 62×62 area around the altered point, with 88% of the affected points concentrated in the center 30×30 area.

parallelization of the algorithm rather than optimizing the compression method. The primary difficulty for using *only* irregular data with the patch method is ensuring that every patch contains enough data and sufficiently overlaps the data from neighboring patches to accurately reconstruct the unknown data points. We leave this as an area for future work.

3. OUR APPROACH

The time it takes to calculate ODETLAP on an elevation map does not scale as the size of the data set increases, and the execution time quickly becomes prohibitively long. Figure 5 illustrates the ODETLAP performance issue. When the terrain is of size 50×50 and 100×100, it takes less than one second to run ODETLAP. However, when we run ODETLAP on an 800×800 terrain, it takes nearly 10 minutes to finish, and 800×800 is a very modest terrain size. Thus, our goal is to develop a scalable implementation of ODETLAP to allow manipulation of large terrains, such as those collected by LIDAR scanners.

As the data set size grows, more and more points are involved in the calculation, some of which are quite distant and thus have little influence on each other. In Figure 6, we show that when we edit a single known point in the input to ODETLAP, only points within a small neighborhood of the point are affected. Beyond that small region, the effect becomes negligible. This supports our hypothesis that it should be possible to divide large data sets into separate patches, run ODETLAP on them individually, and achieve similar results to the non-patch ODETLAP solution. Specifically, a patch size of 100×100 should be sufficient to capture the detail for a terrain grid with a subsampling percentage of 1%. When using a patch size of 100×100, each patch will overlap with a window of 100×50 with its neighbors to the north and south, 50×100 with its neighbors to the east and west, and 25×25 with its diagonal neighbors. If we used a higher subsampling percentage, then less overlap would be needed, so a smaller patch size would be appropriate. When the reconstructed patches are merged back together into the reconstructed elevation map as shown in Figure 7, the difference between the reconstructed and original should

be minimal.

3.1 Dividing into Patches

The ODETLAP calculation depends on elevation data from neighboring pixels, which causes a problem when running it on individual patches. At the edges of a patch, there is less information to work with, and therefore the calculations tend to have errors when compared to the non-patch-based ODETLAP solution. Figure 8 highlights the problem by showing that when the image is reconstructed, the values are generally correct near the center of the patch, but near the edges of the patch, it is common to encounter errors of 5 meters or more.

To determine the value for the error, we calculate ODETLAP on the heightmap using a single large patch that covers the entire elevation map, and then we calculate ODETLAP on each individual patch. The error is defined as the absolute value of the difference between the corresponding elevation values for each method. If we were to simply divide a terrain map into a single layer of non-overlapping patches, then reconstruct them with ODETLAP, and join them at their edges to form the reconstructed image, then we would see results like the ones in Figure 9. At the patch borders, there are areas of large error and drastic discontinuities.

To avoid errors at the borders of the patches, we run ODETLAP on overlapping layers of patches. Figure 2 illustrates an example where instead of dividing an elevation map into 16 non-overlapping patches, we divide the elevation map into 49 overlapping patches.
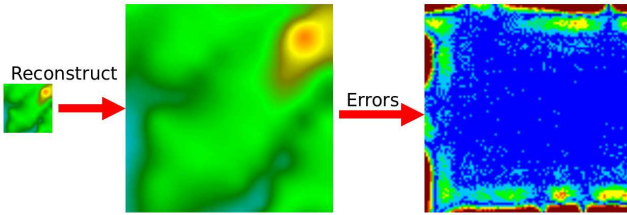
3.2 Merging the Patches
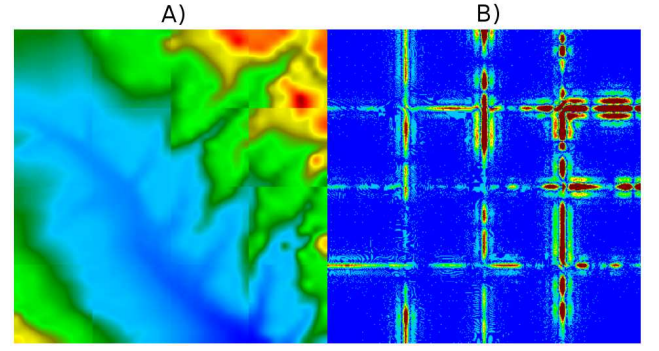
After ODETLAP is calculated on multiple patches, the
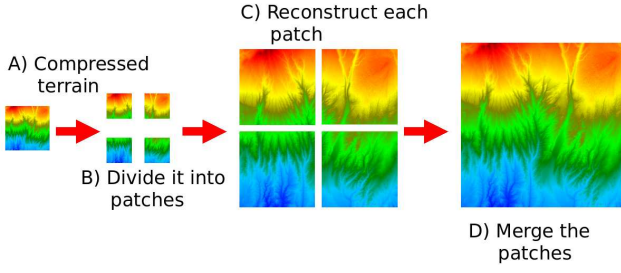
Figure 7: In the patch ODETLAP method, we take the compressed terrain (A), and divide it into patches (B). Next, we run ODETLAP on each patch individually, which reconstructs a small portion of the entire elevation map (C). Finally, we merge all of the patches into the final approximated solution (D).
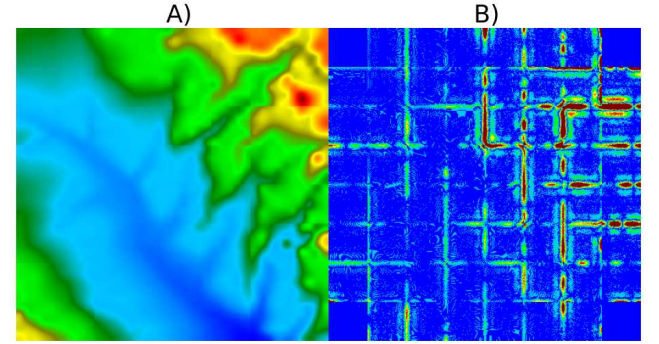

Figure 8: Due to incomplete data, ODETLAP results in errors near the borders of a patch when we compare the results from the non-patch ODETLAP method with the results from running ODETLAP on a small patch. The error plot on the right shows correct results in blue, and elevation differences of 5 or more units (meters) in red.

Figure 9: We get poor results if we naively merge the patches. A) shows discontinuities in the naively merged elevation map, and B) shows the errors.

Figure 10: A simple averaging of overlapping patches reduces some of the border error of the reconstructed terrain. A) shows the image that has been merged by averaging, and B) shows the errors. Notice the visible discontinuities where patch edges are averaged.

patches are merged into the final result. The basic idea is shown in Figure 3. There are many possible ways to go about merging the patches. The overly simple method of uniformly averaging the values of all contributing patches will still contain discontinuities and large errors at the patch edges as seen in Figure 10. However, because we are running ODETLAP redundantly for each pixel, we have the option to ignore the erroneous edge pixels by selecting those values from the center of a different patch.

In order to accomplish this, we use bilinear interpolation which weights pixels near the center very strongly, and the weight falls off to zero for pixels near the edges. A visualization of the weights can be seen in Figure 11. The image on the left shows the bilinear interpolation weighting pattern for a single patch. The solid green image on the right shows the weighting pattern using simple averaging. Blue pixels represent a weight of 0.0, and red pixels represent a weight of 1.0. In both weighting methods, when four patches are merged, the sum of all of the weights for a given pixel is one. Using bilinear interpolation, we are able to merge the patches such that the pixels at the patch's borders are ignored, but the correct pixels near the center contribute most of the value. This results in a reconstructed elevation map that has very small errors, and no visible discontinuities, as seen in Figure 12.

Because non-patch ODETLAP is prohibitively slow on large data sets, the error analysis and plots shown in Figures 8, 9, 10, and 12 were performed on 400×400 terrain data. The elevations for this DEM range from 1105 to 1610 meters, and are represented as integer values with units of one meter.

4. IMPLEMENTATION

We implemented the patch version of ODETLAP using MPI, which allows us to run the software on parallel and distributed platforms. When MPI starts, the first process is assigned the task of waiting for reconstructed patch data from the rest of the processes, which are designated as *worker* processes. All of the workers are pre-assigned a set of patches to process. For each assigned patch, the process does the following:

- Load the patch

- Run ODETLAP on the patch

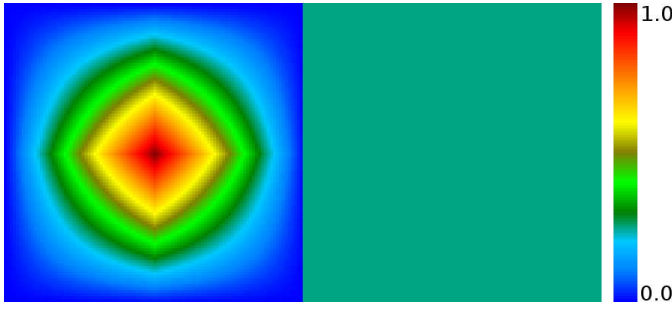- Send the reconstructed patch to the central process

Figure 11: We use bilinear interpolation (left) instead of a simple averaging (right) to do a weighted averaging of four pixels to merge four patches. Note that the corners and edges form special cases where only one and two patches contribute to the result.
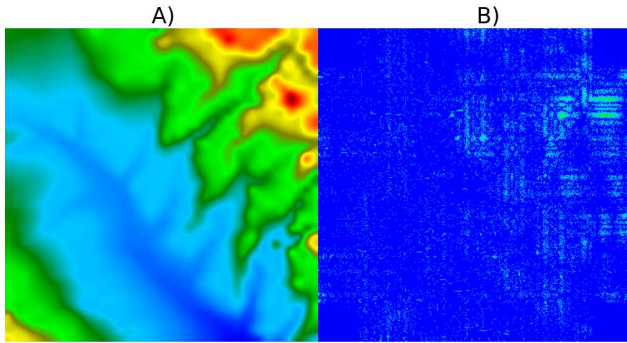

Figure 12: We use bilinear interpolation to do a weighted average such that border values fall off to zero. This results in a visibly continuous reconstructed image (A), and small error values (B), when compared with results from running the non-patch version of ODETLAP on the same terrain.

As the patches are collected by the first process, the values are weighted and merged into the full reconstructed elevation map, which is then saved to the hard disk.

A simple direct linear system solver would have required cubic time to execute. However, in our implementation we built ODETLAP on the QR solver from the CSparse library [3]. We have found the execution time to be nearly quadratic with respect to the number of points in a single patch, and in the case of the non-patch ODETLAP version, with respect to the number of points in the entire input elevation map.

We implemented the parallelized patch ODETLAP on a cluster of 2.6 GHz AMD Opteron machines running Red Hat Enterprise Linux 4.5. We have also tested the program on the Blue Gene/L supercomputer. The Blue Gene presents additional challenges because each processor can optimally access only 512MB of memory. Also, running ODETLAP on a very large number of processors requires extra care to ensure that the slowness of disk access does not cause bottlenecks that prevent the worker processes from working at full capacity. We will discuss the Blue Gene further in Section 6.
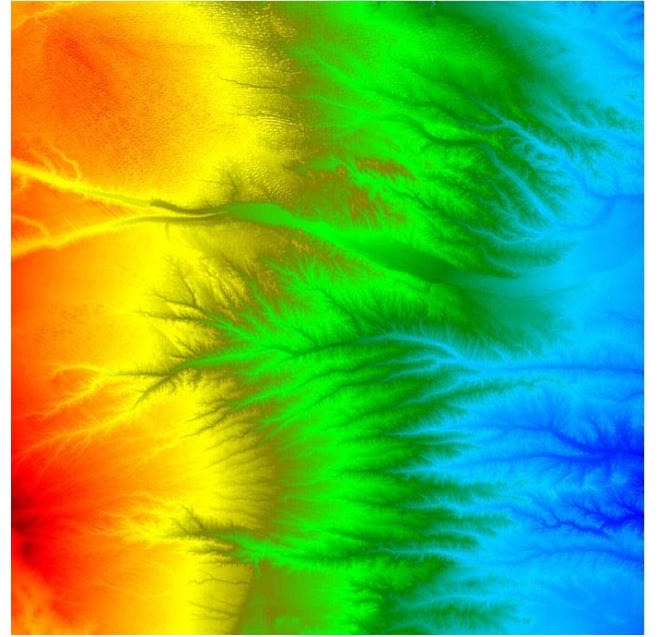

Figure 13: This 16000×16000 elevation matrix can be processed by parallelized ODETLAP in 28 minutes and 32 seconds.

5. RESULTS

We tested our parallelized ODETLAP on a 16000×16000 DTED Level 2 data set covering roughly 400,000 square miles primarily occupied by the states of Kansas and Nebraska in the USA. The terrain is divided into 101761 patches, each with a resolution of 100×100. We used 128 processors on a cluster of 2.6 GHz AMD Opterons and the entire computation took 28 minutes and 32 seconds. The full terrain is shown in Figure 13. The range (highest minus lowest) of the test data is 1013 meters and the standard deviation is 217 meters. We select every 10th sample point in both the x and y dimensions to generate a set of input points consisting of 1% of the total points. A single 100×100 patch consisting of 10000 points is represented by 100 points.

Compared to the original terrain, the reconstructed terrain has mean absolute error of 1.96, max absolute error of 50, and root mean square error of 2.76. Note that it is impossible to run the non-patch version of ODETLAP on this large data set, so for these results we are comparing to the original. A comparison of the original and reconstructed terrain is given in Figures 14, 15, and 16. The two images in Figure 14 correspond to the 1000×1000 patch in the top left corner of the original terrain in Figure 13. We can see in the figure that the reconstructed terrain is only losing some high frequency details. A difference map between the original and the result (Figure 15) shows which areas perform well, and where smoothing occurs. Figure 16 presents a close-up view of the area with the highest error, taken from the oxbow river flowing from the southeast corner to the center of Figure 13. The largest errors occur due to the presence of many extreme variations in elevation within a small (100×100) area. Many of these details are captured with the more advanced point selection schemes described
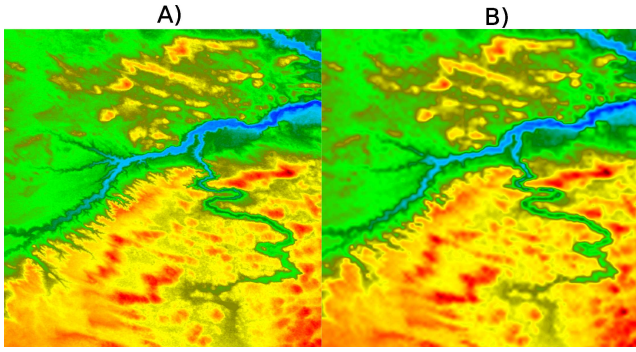
Figure 14: The original terrain (A) is compared to terrain that has been compressed by sampling on a regular grid, and then reconstructed using the patch method (B). This terrain corresponds to the northwestern corner of Figure 13. This portion of the elevation map has a range of 163 meters. Notice that some smoothing has occurred.
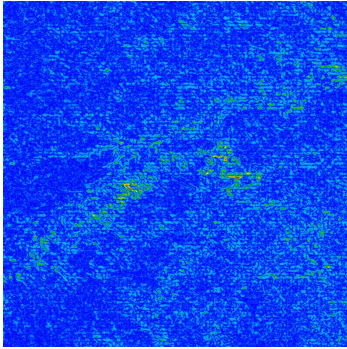

Figure 15: A difference map that shows the error between (A) and (B) from Figure 14. The error has a range of 0..29 meters.
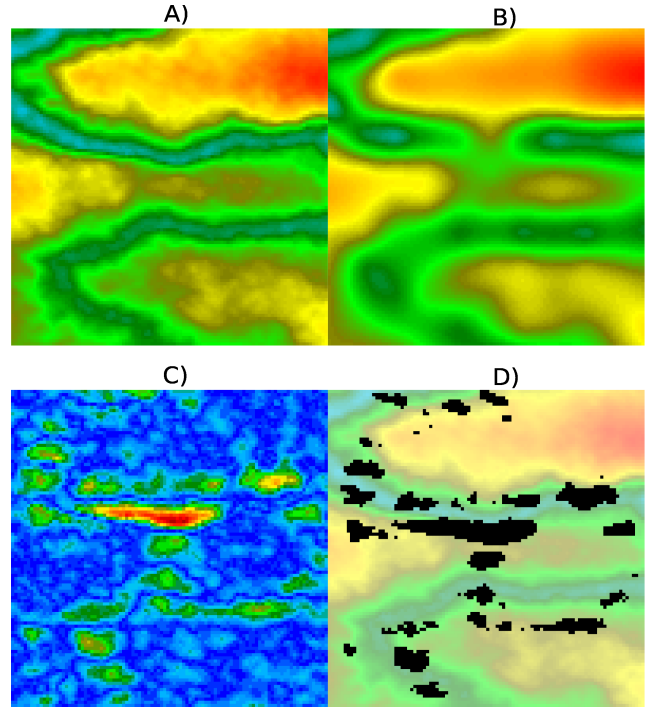

Figure 16: (A) and (B) show a detailed view of a particularly challenging 100×100 section taken from the grid in Figure 14. (C) shows a difference map with a range of 0 (blue) to 29 (red) meters. (D) shows the areas with an error larger than 10 in black.

in [13], or when a compression by less than a factor of 100 is used.

We also investigate the impact of using different patch sizes on both running time and reconstruction accuracy. In Table 2, we use patch sizes from 20×20 to 400×400, and record the running time, mean absolute error, maximum error, and RMS error of the reconstruction. We can see that using a patch size of 100×100 gives a nice balance between accuracy and speed. Table 3 shows the small amount of error when comparing the patch version of ODETLAP to the non-patch version. By examining the amount of error introduced versus the overall speedup that is gained, using patches that are 100×100 in size produces very good results.

The patch-based ODETLAP algorithm provides two performance improvements. The first is from decomposing large-scale terrain data into small patches and running ODETLAP sequentially on each of them, which we call *serialized ODET-LAP*. Figure 17 shows the running time comparison of the non-patch version of ODETLAP and serialized patch-based ODETLAP. From the figure, we can see that the running time is greatly reduced even if no parallelism is used.

The second speedup occurs because we are running ODET-

LAP on multiple patches concurrently in a parallel environment. In Figure 18 we show the total running time for a test case with 1521 patches when run on various numbers of processors. We see that parallelism provides an excellent speedup using up to 127 worker processes. Beyond that, the overhead of parallelism becomes significant. We have opportunities to improve the performance even further. We will need to look closely at ways to optimize disk I/O, which is currently our primary bottleneck. We will discuss these improvements in more detail in Section 6.

In Table 4, we present the running time and accuracy information for all three ODETLAP versions. The size of the input terrain data is 800×800, and the mean elevation is 107. We can see that the patch method only increases errors by approximately 0.1% when compared with the non-patch method, and the running time is reduced to about 0.2% of the original.

6. FUTURE AND ONGOING WORK

In our implementation, the size of the input elevation map is limited by the the amount of memory required to store the entire grid while merging it. In order to overcome this limitation, we are working on streaming the output to the disk as it is merged and freeing the memory for completed patches. Also, the implementation has a bottleneck when reading the input data from disk. The input data set is loaded once for every patch that needs to be calculated. For example, when calculating ODETLAP on a 16000×16000 grid, there are

Patch Size	Time	Mean Error	Max Error	RMS Error
20 × 20	0m14s	0.6570	13	0.9798
40 × 40	0m7s	0.6619	13	0.9594
50 × 50	0m8s	0.6640	13	0.9617
100 × 100	0m9s	0.6598	13	0.9530
200 × 200	0m25s	0.6598	13	0.9527
400 × 400	1m28s	0.6598	13	0.9527

Table 2: **Comparison of different patch sizes: Starting with original terrain of size** 2000×2000, **input points are sampled every 10 points in the x and y directions, thereby selecting a total of** 1% **of the total points. A patch size of** 100×100 **gives a good compromise between running time and accuracy. Reported errors in these examples are with respect to the original terrain data set.**
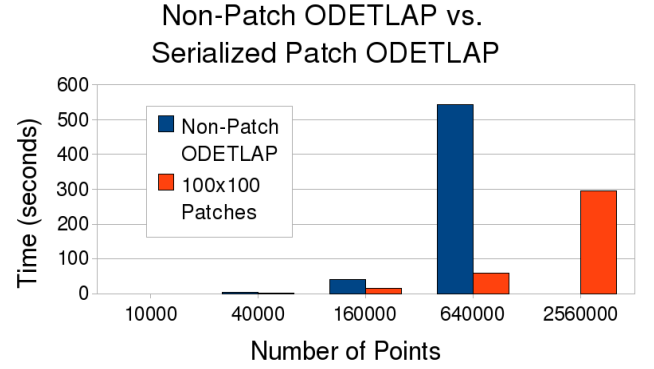
Patch Size	Mean Error	Max Error	RMS Error
20 × 20	0.1431	2	0.3801
40 × 40	0.0532	1	0.2307
50 × 50	0.0634	2	0.2519
100 × 100	0.0149	1	0.1223
200 × 200	0.0039	1	0.0628
400 × 400	0.0008	1	0.0291

Table 3: **This companion data to Table 2 reports the errors of Patch ODETLAP with respect to Non-Patch ODETLAP processing of the same test data consisting of** 800×800 **points. Note that the parallel version introduces only a very small amount of error.**

Figure 17: **Execution time for the non-patch version of ODETLAP grows quadratically with the number of pixels, while the patch version of ODETLAP has a linear growth. This improvement is gained while running serially.**

Method	Time	Mean Err.	Max Err.	RMS Err.
Non-P ODETLAP	9m8s	0.6150	7	0.8835
Seri-ODETLAP	0m34s	0.6156	7	0.8846
Patch ODETLAP	0m2s	0.6156	7	0.8846

Table 4: **Comparison of three versions of ODET-LAP: non-patch, serialized, and parallel on an 800x800 data set. Reported errors are with respect to the original terrain data set.**

101761 patches, and the file is loaded once for each of those patches. When we use more processors, it means that more processors are all trying to read the file simultaneously. This results in limited scalability. We have designed the next version so that the input only needs to be accessed by a single source process. The source process distributes the data to the worker processes, thereby alleviating the bottleneck. A further improvement would be to allocate multiple source processes for larger data sets, allowing us to run ODETLAP using any number of available processes efficiently.

The patch method described in this paper creates an artificial coupling of patch size with the amount of overlap needed for accurate results. The algorithm would gain flexibility if a patch represented an independent square of data that did not overlap with any other data patches, and the amount of overlap were represented separately. For example, one could specify a patch size of 50×50 with an overlap of 25 pixels in each direction. ODETLAP would then be calculated on 100×100 blocks except for patches that lie in the edges and corners of the data set. Abstracting the overlap this way is described as a "halo" in [6]. The patch size would be selected to optimize speed, and the amount of overlap would be selected to ensure an acceptable accuracy in the results. Implementing the patch method in this way would lend itself to a multigrid implementation and to striding the data set across multiple processes.

As shown in Figure 12, there are still some errors compared with results from the non-patch version of ODET-

LAP, which means that we may still be able to improve the patch merging process to get higher accuracy. We would need to analyze the results and find out what causes the errors, and choose the most appropriate interpolation scheme based on our findings. This can be combined with strategies to optimize for elevation maps with irregular sampling such as point cloud data or Triangulated Irregular Network (TIN) data. For example, patches with more points should be weighted more heavily than undersampled patches when doing interpolation in the overlapped regions. Additionally, multigrid could be used as a point selection strategy in which high frequency areas are more densely sampled than flat areas. Integrating a multigrid approach with the parallel patch method would lead to interesting research.

We have performed initial test runs on the 32,768 processor Blue Gene/L system that is part of Rensselaer's Computational Center for Nanotechnology Innovations (CCNI). With 32,768 processors, the Blue Gene/L system will enable us to get much faster performance than what could otherwise be possible. The Blue Gene presents a challenge because each processor is limited to 512 MB of memory in its optimal configuration. For this reason, we have designed two running modes for the Blue Gene. The first mode is optimized for data sets that are small enough to fit into a single process' memory. In this mode, the data is read from disk, distributed to the workers, and merged in the sink before being written to the disk. This gives the fastest execution time, but will not run for very large data sets (16000×16000 and larger). The second mode trades the speed benefits of the first method for nearly limitless data set sizes by us-
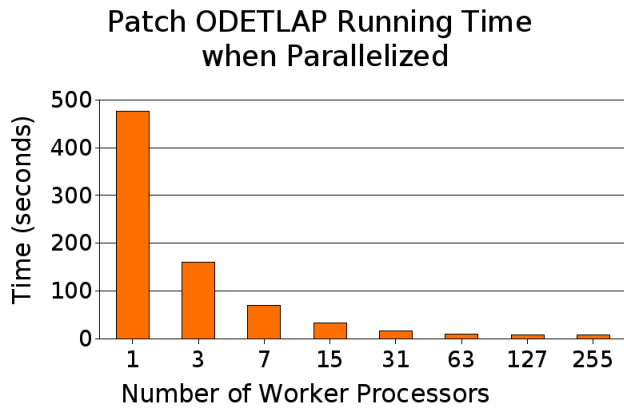
Figure 18: We reconstruct a 2000×2000 **grid from a subset of** 1% **of the original points by running ODET-LAP on a distributed platform consisting of a cluster of AMD Opterons running at 2.6GHz. We were able to achieve a linear decrease in running time for up to 128 processors before the overhead of file I/O prevents us from gaining any additional speedup without further optimizations. The algorithm includes a central process to merge results, but this process is not included in the x axis.**

ing the disk as a cache. This method is quite a bit slower than the first. In order to get the best of both worlds, we would need to allocate processes whose only job is to act as a cache for the merging process. Inter-process communication on the Blue Gene is incredibly fast compared to the speed of accessing the file system, so by avoiding disk access as much as possible we would enable our implementation to run ODETLAP on huge data sets quickly.

7. CONCLUSIONS

In this paper we have presented our recent progress in parallel terrain compression and reconstruction that processes digital elevation maps of sizes as large as 16000×16000. We use ODETLAP to reconstruct a terrain map from sparse, isolated samples. In order to greatly increase the efficiency of ODETLAP, we divide the original terrain into patches which are then run on multiple processors in parallel. A patch size of 100×100 provided a very good performance gain, while minimally impacting the results of output when compared to the non-patch version of ODETLAP. The results from experiments show that our method greatly reduces the running time and ensures a high quality in the reconstructed image. This new technique changes the way that we will approach large terrains in the future.

8. ACKNOWLEDGMENTS

9. REFERENCES

[1] A. Clematis, B. Falcidieno, and M. Spagnuolo. Parallel Processing on Heterogeneous Networks for GIS Applications. *International Journal of Geographical Information Science*, 10(6):747–767, 1996.

[2] A. Clematis, M. J. Mineter, and R. Marciano. High performance computing with geographical data. *Parallel Computing*, 29(10):1275–1279, 2003.

[3] T. A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[4] W. R. Franklin, M. Inanc, Z. Xie, D. M. Tracy, B. Cutler, M. V. A. Andrade, and F. Luk. Smugglers and border guards - the geostar project at rpi. In *15th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS 2007)*, Seattle, WA, USA, Nov 2007.

[5] M. Griebel and G. Zumbusch. Parallel multigrid in an adaptive pde solver based on hashing and space-filling curves. *Parallel Comput.*, 25(7):827–843, 1999.

[6] R. G. Healey, M. J. Minetar, and S. Dowers, editors. *Parallel Processing Algorithms for GIS*. Taylor & Francis, Inc., Bristol, PA, USA, 1997.

[7] F. Huang, D. Liu, P. Liu, S. Wang, Y. Zeng, G. Li, W. Yu, J. Wang, L. Zhao, and L. Pang. Research On Cluster-Based Parallel GIS with the Example of Parallelization on GRASS GIS. In *GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing*, pages 642–649, Washington, DC, USA, 2007. IEEE Computer Society.

[8] M. Kazhdan and H. Hoppe. Streaming multigrid for gradient-domain operations on large images. *ACM Transactions on Graphics*, 27(3):21:1–21:10, Aug. 2008.

[9] M. J. Mineter. A software framework to create vector-topology in parallel GIS operations. *International Journal of Geographical Information Science*, 17(3):203–222, 2003.

[10] J. E. Mower. Developing Parallel Procedures for Line Simplification. *International Journal of Geographical Information Science*, 10(6):699–712, 1996.

[11] S. C. Roche and B. M. Gittings. Parallel Polygon Line Shading: The Quest for More Computational Power from an Existing GIS Algorithm. *International Journal of Geographical Information Science*, 10(6):731–746, 1996.

[12] Y. F. Shuichi Ichikawa. Iterative Data Partitioning Scheme of Parallel PDE Solver for Heterogeneous Computing Cluster. In *Proceedings of IASTED Int'l Conf. Applied Informatics: Int'l Symp. Parallel and Distributed Computing and Networks*, pages 364–369, Seattle, WA, USA, Nov. 2002. ACTA Press.

[13] Z. Xie, W. R. Franklin, B. Cutler, M. A. Andrade, M. Inanc, and D. M. Tracy. Surface compression using over-determined laplacian approximation. In *Proceedings of SPIE Vol. 6697 Advanced Signal Processing Algorithms, Architectures, and Implementations XVII*, San Diego CA, 27 August 2007. International Society for Optical Engineering. paper 6697-15.