Proc. 3rd IEEE International Conference on e-Science and Grid Computing,
Bangalore, India, December, 2007, pp. 337-344

Distributed and Generic Maximum Likelihood Evaluation

Travis Desell† Nathan Cole‡ Malik Magdon-Ismail† Heidi Newberg‡
Boleslaw Szymanski† Carlos Varela†

†Department of Computer Science ‡Department of Physics, Applied Physics, and Astronomy
Rensselaer Polytechnic Institute, Troy, NY, U.S.A.
†{deselt, magdon, szymansk, cvarela}@cs.rpi.edu ‡{colen2, newbeh}@rpi.edu

Abstract

This paper presents GMLE [1], a generic and distributed framework for maximum likelihood evaluation. GMLE is currently being applied to astroinformatics for determining the shape of star streams in the Milky Way galaxy, and to particle physics in a search for theory-predicted but yet unobserved sub-atomic particles. GMLE is designed to enable parallel and distributed executions on platforms ranging from supercomputers and high-performance homogeneous computing clusters to more heterogeneous Grid and Internet computing environments. GMLE's modular implementation seperates concerns of developers into the distributed evaluation frameworks, scientific models, and search methods, which interact through a simple API. This allows us to compare the benefits and drawbacks of different scientific models using different search methods on different computing environments. We describe and compare the performance of two implementations of the GMLE framework: an MPI version that more effectively uses homogeneous environments such as IBM's BlueGene, and a SALSA version that more easily accommodates heterogeneous environments such as the Rensselaer Grid. We have shown GMLE to scale well in terms of computation as well as communication over a wide range of environments. We expect that scientific computing frameworks, such as GMLE, will help bridge the gap between scientists needing to analyze ever larger amounts of data and ever more complex distributed computing environments.

[1]This work has been partially supported by the following grants: NSF AST No. 0607618, NSF IIS No. 0612213, NSF MRI No. 0420703 and NSF CAREER CNS Award No. 0448407. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

For many scientific disciplines, the rate of data acquisition far exceeds the increases in computational power required to analyze and mine valuable information from the data. A wide range of scientific disciplines use maximum likelihood evaluation (MLE) methods, and as Cramer stated: *"From a theoretical point of view, the most important general method of estimation known so far is the method of maximum likelihood"*[8]. MLE attempts to find global optimal parameters for a certain function to fit a given data set. There are many different approaches to performing this task, such as gradient descent [22], simplex [16], and genetic search [10], all of which typically require hundreds, thousands or more function evaluations to converge to a result within a reasonable margin of error. Since each function evaluation can require hours or more to complete on a single high-end processor, performing MLE can result in month-long single-CPU execution times—to test a single function.

A distributed framework for MLE makes data analyses more tractable by streamlining the scientific cycle, reducing the process of testing new models and functions from days to hours. A generic framework makes this capability accessible to scientific researchers who may not have the resources to develop their own distributed computing framework. The genericity of the framework also provides the opportunity to evaluate multiple search methods and use the ones which provide the best performance for a given application. To this end, we have leveraged our previous work on MLE applied to particle physics [25] to make it generic, allowing for scientific functions and data sets, the distributed evaluation framework, and different search methods to be developed concurrently and independently.

To test the genericity of the framework, we have implemented an astroinformatics application to discover star streams structure from observations of the Milky Way in addition to a particle physics application to discover missing

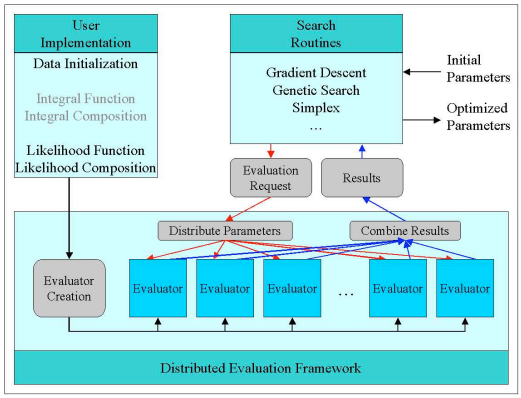

Figure 1. A generic framework for distributed maximum likelihood evaluation.

baryons. We have also extended the simplex search method available in the previously reported framework with conjugate gradient descent and genetic search methods. We have implemented two versions of GMLE, one using the MPI message passing library and another one using the SALSA programming language [24]. We examine the performance of the SALSA and MPI implementations on a homogeneous cluster in addition to the performance and speedup of using the SALSA implementation on the Rensselaer Grid and the MPI implementation on an IBM BlueGene supercomputer.

Structure of the Paper Section 2 describes the GMLE framework and its components. The astroinformatics and physics applications currently using the framework are presented in Section 3. How the different search methods apply to distributed environments is discussed in Section 4. The APIs for developing scientific functions for use in the SALSA and MPI implementations are discussed in Section 5, and simple examples are given for each. Section 6 presents empirical results. Related work is discussed in Section 7 and we conclude with future work and conclusions in Section 8.

2 Generic Framework for Distributed MLE

The GMLE package presented in this paper consists of three components (see Figure 1): 1) A user-defined function and a *combiner*, a partial result composition operation which specify the problem to which MLE is applied, 2) A distributed evaluation framework which initializes the function and combiner and performs distributed evaluations and 3) Search methods which perform the MLE.

Each of these components requires different expertise: domain knowledge, distributed computing, and machine learning respectively. However, these components interact with simple interfaces (for more detail, see Section 5) which allow them to be developed independently. Hence, each expert involved in the development can focus on their respective area. This solution provides multiple scientific users, such as those in astroinformatics [19] or physics [25], an easy to use package for distributing their MLE programs without significant development effort and directly benefiting from performance improvements in non-functional framework modules. It also allows for research into different methods of distributed function evaluation and distributed MLE searches.

In the GMLE framework, users define a likelihood function on which the MLE will be performed. This is done by creating an *evaluator* which initializes data and performs the likelihood operation on this data based on a set of parameters. Because likelihood evaluation is distributed over multiple evaluators, a result composition method also needs to be specified that combines partial results into the final likelihood value. In some cases, such as the astronomy application described in Section 3.1, an expensive integral needs to be calculated before the likelihood evaluation, so there is also an option to create an integral function whose calculation can be distributed by the evaluation framework in the same way and then used to calculate the likelihood.

After the likelihood, integral, initialization and composition methods are specified, a user runs a search method with initial function parameters and other search method parameters, if necessary. The distributed evaluation framework initializes evaluators at each processor being used by the system, and the search method can then repeatedly request evaluations for different function parameter sets.

Two different distributed evaluation methods can be used, an asynchronous master/worker (or farmer/worker) method or a synchronous parallel slice method. The synchronous parallel slice method gives the evaluator at each processor an equal slice of the data, evaluates the likelihood at each evaluator concurrently, then combines the result. This approach is best for iterative searching methods such as gradient descent and simplex, and for homogeneous tightly-connected computing resources.

The asynchronous master/worker method creates evaluators which operate over the total data set and perform a complete likelihood evaluation. In this case, the composition is not required, and results from individual evaluators can be passed directly back to the search routines. This allows multiple parameter sets to be evaluated concurrently. This approach is better suited to search methods which allow for parallelism in likelihood evaluation or are non-iterative, such as genetic search and other Monte Carlo methods. Because parameter sets are evaluated concurrently and asynchronously, faster processors can evaluate the likelihood for multiple parameter sets without being delayed by slower processors, making this approach better suited to heterogeneous loosely-connected computing envi-

ronments.

3 Scientific Applications

The two applications currently implemented using GMLE are representative of those that analyze the large quantities of data required for modern science. In both cases, a single function evaluation can take hours on a single processor, and days or weeks to perform a single maximum likelihood fit.

3.1 Origin and Structure of the Milky Way Galaxy

The astronomy application attempts to discover various structures existing in the Milky Way galaxy and their spatial distribution [19]. This requires a probabilistic algorithm for locating geometric objects in spatial databases [20]. The data being analyzed is from the Sloan Digital Sky Survey [1], which has recently released 10 terabytes of images and spectra in online catalogs.

The observed density of stars is drawn from a mixture distribution parameterized by the fraction of local substructure stars in the data compared to a smooth global background population, the parameters that specify the position and distribution of the sub-structure, and the parameters of the smooth background. Specifically this is a probability density function (PDF) that calculates the chance of obtaining the observed star distribution after repeated independent sampling from the total set of stars:

$$\mathcal{L}(\vec{Q}) = \prod_{i=1}^{N} PDF(l_i, b_i, g_i|\vec{Q}) \quad (1)$$

where i denotes the i^{th} of N stars (l and b are galactic coordinates and g is the magnitude), and $\vec{Q}$ represents the parameters in the model. Such probabilistic framework gives a natural sampling algorithm for separating sub-structure from background [19].

By identifying and quantifying the stellar substructure and the smooth portion of the Milky Way's spheroid, it will be possible to test models for the formation of our galaxy, and by example the process of galaxy formation in general. In particular, we would like to know how many merger events contributed to the build up of the spheroid, what the sizes of the merged galaxies were, and at what time in the history of the Milky Way the merger events occurred. Models for tidal disruption of merger events that build up the spheroid of the Milky Way can be matched with individual, quantified spatial substructures to constrain the Galaxy's gravitational potential. Since the gravitational potential is dominated by dark matter, this technique will also teach us about the spatial distribution of dark matter in the Milky Way.

3.2 Searching for Missing Baryons

The physics application uses Partial Wave Analysis (PWA) [25] to observe particle states and measure their quantum numbers known as "spin" and "parity". One way to measure this is by producing a beam of mesons (pions for example) in an accelerator and striking a liquid hydrogen target with that beam. Some fraction of these pions interact with a proton at the target, and if the energy of the pion is high enough, a spray of particles is produced. Some of the particles will live long enough to create trails in a suitable detector, but many will decay after an extremely short time (10^{-23} seconds) and will not travel a measurable distance. These short lived particles are the "missing" baryons.

The existence and properties of missing baryons can be inferred from correlations in the final state particles into which they decay. The measured final or intermediate states of these particles produce the data set and PWA calculates the amount of each spin-parity state present in this data set. Maximum likelihood is used to find the set of parameters which define the most likely set of amplitudes for producing each of the observed intermediate states:

$$-\ln(\mathcal{L}) = -\sum_{i}^{n} \ln\left(\left|\psi_\alpha^p \psi_\alpha^d(\tau_i)\right|^2\right) - n\psi_\alpha^p \Psi_{\alpha\alpha'} \psi_{\alpha'}^{p*} \quad (2)$$

where the sum over i runs over all events in the data set, and the sums over the repeated αs, the fit parameter index, are implicit. The ψ_α^p are the complex fit parameters, related to the amount of the intermediate state α produced. The $\psi_\alpha^d(\tau_i)$ is the quantum amplitude for the i^{th} event with angles τ_i assuming intermediate state α. The second term on the right hand side is the normalization integral, where any known inefficiencies of the detector are taken into account. The total number of events in the data being fit is n; and $\Psi_{\alpha\alpha'}$ is the result of the normalization integral, done numerically before the fit is performed. Currently, there are 10-100 fit parameters and around 10^5 events, however these numbers are increasing and will approach 10^6 or 10^7 with the next generation of particle detectors.

4 Search Methods

Gradient Descent Gradient descent takes an initial parameter set and calculates the gradient at that point by calculating the slope for each parameter using a specified step size. The slopes of all parameters are used to calculate a gradient, along which a line search is performed to find the minima of the function on that line. This process repeats until the gradient descent finds a local minimum. The GMLE

package also supports conjugate gradient descent, which is an improvement on gradient descent. Instead of calculating the gradient, it calculates the conjugate gradient which is a modification of the gradient that frequently results in faster convergence to the local minimum.

Simplex The simplex method used by the GMLE package is the Nelder and Mead or downhill simplex method [16]. $N + 1$ initial points of the function are calculated by $N + 1$ parameter sets, which define the initial simplex. The downhill simplex method then takes the highest point of the simplex, and evaluates a new point reflected through the simplex. The algorithm then either does a line search by continuing to expand the reflected point farther away from the simplex (if this continues to find a lower point), or closer to the simplex (if that results in a lower point). After a minima for the new reflected point is found, the algorithm repeats until some cutoff conditions are reached and a local minimum is found.

Genetic Search Genetic search generates an initial random population of parameter sets. Given a population, the algorithm evolves that population by generating new parameter sets through mutation, reproduction, and elimination. Parameter sets have a chance to mutate, creating a new parameter set with a random change that is then evaluated and added to the population. Two parameter sets can reproduce to create an offspring, which is a combination of the two. Typically, a fixed population is kept, so when new parameter sets are evaluated, the ones with the lowest likelihood are eliminated. Mutation allows a genetic search to branch out with new parameter sets which might find another optimum, and reproduction allows the parameter sets around an optimum to converge to that point.

Tradeoffs and Distribution While the likelihood evaluations required to calculate the gradient or simplex can be done concurrently, the line search computes a new parameter set iteratively based on previously calculated parameter sets. Additionally, calculating a new gradient or simplex reflection requires the previous line search to have finished. This means that any type of gradient descent or simplex can only scale as far as an individual distributed likelihood evaluation. Additionally, the minimum found is a single minimum local to the initial parameters. However, these disadvantages are offset by the fact that gradient descent typically converges to a minimum faster than genetic search. Both simplex and gradient descent quickly converge to a local minimum, however which will converge to a minimum the fastest is typically application dependent.

Genetic search, unlike the gradient descent and simplex methods, does not require an iterative evaluation of the likelihood function. This allows for multiple likelihood evaluations to be performed concurrently, which can improve the scalability of the application. For applications with faster likelihood evaluation times and that have access to larger numbers of processors, a genetic search may be able to converge faster than gradient descent or simplex. Additionally, genetic search has the benefit of having sub-populations converge to multiple minima, which can be valuable information. Gradient descent and simplex require multiple runs of the algorithm with different starting conditions to find different minima.

```
#include "gmle/framework/gradient_descent.h"
char* FILE = "constants.txt";
double* constants;
int total_constants = 100;
int min, max, length;
void init_data(int rank, int max_rank) {
  min = total_constants*rank/max_rank;
  max = total_constants*(rank+1)/max_rank;
  length = max-min;
  //read all constants from FILE
  //constants[0..length] = constants[min..max]
}
double likelihood(double* parameters) {
  double sum = 0.0;
  for (int i = min; i < max; i++)
    sum += constants[i-min] * parameters[i] * parameters[i];
  return sum;
}
double combine(double* results, int n_results) {
  double sum = 0.0;
  for (int i = 0; i < n_results; i++) sum += results[i];
  return sum;
}
int main(int argc, char** argv) {
  int size = sizeof(double)*total_constants;
  double* initial_parameters = malloc(size);
  double* step_sizes = malloc(size);
  for (i = 0; i < total_constants; i++) {
    initial_parameters[i] = drand48() * 10;
    step_sizes[i] = 0.000001;
  }
  conjugate_gradient_descent__init_likelihood(likelihood,
    combine);
  conjugate_gradient_descent(initial_parameters,
    step_sizes,length);
}
```

Figure 2. A simple sum of squares example using the MPI package (file I/O replaced with pseudo-code).

5 Implementation

The GMLE framework is implemented in MPI as well as SALSA, to allow for interoperability with different types of legacy code—C code in the case of MPI, and Java code in the case of SALSA. The MPI implementation allows the code to be run on supercomputing architectures such as the IBM BlueGene as well as high performance cluster environments with minimal communication overhead. The SALSA implementation allows for easy use of the framework on heterogeneous environments by leveraging the Java Virtual Machine [24] and transparent autonomous reconfiguration provided by IOS [13], the Internet Operating System.

Using the MPI and SALSA implementations involve similar APIs [2]. Figures 2 and 3 demonstrate usage of the GMLE package with a simple sum of squares example. Both examples demonstrate the use of the conjugate gradient descent method. Note that for both the details of remote communication, such as MPI function calls and Java

[2]The APIs, documentation, more in depth examples and details about using the search methods can be found at http://wcl.cs.rpi.edu/gmle/

```
import gmle.salsa.framework.Evaluator;
import gmle.salsa.framework.ConjGradientDescent;
behavior SumOfSquares implements Evaluator {
  String FILE = "constants.txt", theaters_file = "theaters.txt";
  int total_constants = 100, min, max, length;
  double[] constants;
  void init_data(int rank, int max_rank) {
    min = total_constants*rank/max_rank;
    max = total_constants*(rank+1)/max_rank;
    length = max-min;
    //read all constants from FILE
    //constants[0..length] = constants[min..max]
  }
  double likelihood(double[] parameters) {
    double sum = 0.0;
    for (int i = min; i < max; i++)
      sum += constants[i-min] * parameters[i] * parameters[i];
    return sum;
  }
  double combine(double* results, int n_results) {
    double sum = 0.0;
    for (int i = 0; i < results.length; i++)
      sum += results[i];
    return sum;
  }
  public void act(String[] arguments) {
    double[] initial_parameters = new double[total_constants];
    double[] step_sizes = new double[total_constants];
    for (i = 0; i < total_constants; i++) {
      initial_parameters[i] = drand48() * 10;
      step_sizes[i] = 0.000001;
    }
    ConjGradientDescent cgd = new ConjGradientDescent();
    cgd<-init_data(self.getClass(),theaters_file);
    cgd<-conjugate_gradient_descent(initial_parameters,
      step_sizes);
  }
}
```

Figure 3. A simple sum of squares example using the SALSA package (file I/O replaced with pseudo-code).

sockets are completely transparent to the developer. In the SALSA example `theaters_file` specifies the processors that will run the application.

In the MPI implementation, the methods for reading data, evaluating the function, and combining the results are passed to the distributed evaluation framework via the `conjugate_gradient_descent_init_data` and `conjugate_gradient_descent_init_likelihood` methods. In the SALSA implementation, an actor is defined which implements the evaluator interface and defines the `init_data`, `likelihood` and `combine` message handlers. When the `conjugate_gradient_descent` method is called, the distributed evaluation framework will call the defined `init_data` methods and pass the total number of MPI processes or evaluator actors being created and the rank of that process/evaluator. This will load the data, and then the processes/evaluators will repeatedly perform function evaluations with different input parameters until the search method converges. Data initialization is performed the same way for all search methods, however the simplex and genetic search methods require different initial parameters that define how the search is performed.

6 Empirical Results

The performance of the MPI and SALSA implementations were tested on a homogeneous cluster, a supercomputer and heterogeneous grid. The astronomy application

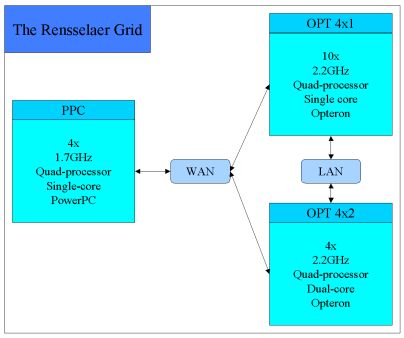

Figure 4. The three clusters in the Rensselaer Grid used for experiments.

was run with a smaller sample data set on each environment to measure the communication overhead incurred by the SALSA and MPI implementations and to show how the framework scales on these different environments.

6.1 Test Environments

The homogeneous cluster (PPC) consists of four quad-processor single-core Power-PC processors running at 1.7GHz with 2GB RAM, for a total of 44GB RAM. Intra-cluster communication on the cluster is provided by 1GB/sec bandwidth with 100μsec latency Ethernet. The PPC processors are running AIX version 5.3 as the operating system.

The heterogeneous grid environment tested uses the Power-PC (PPC) cluster, and two Opteron (OPT) clusters (see Figure 4). The first Opteron cluster (4x2 OPT) consists of 10 quad-processor, dual-core processors, and the second (4x1 OPT) consists of 4 quad-processor single-core processors, all running at 2.2GHz. The dual-core nodes have 32GB RAM, for a total of 128GB RAM, and the single-core nodes have 16GB RAM for another 160GB RAM (192GB in all). The OPT processors are running GNU/Linux version 2.6 as the operating system. Communication within and between the OPT clusters is provided by 10GB/sec bandwidth, 7usec latency Infiniband, and 1GB/sec bandwidth, 100μsec latency Ethernet. The PPC cluster and Opteron clusters are connected over RPI's wide area network forming the Rensselaer Grid testbed.

The supercomputing environment used is RPI's IBM BlueGene/L system. Our experiments used one rack of 1024 nodes, each with two 700MHz Power-PC 440 processors with 1GB RAM, for a total of 1TB RAM across the BlueGene. Inter-node communication is provided by a 3-dimensional torus with 175MBps in each direction, and 1.5usec latency. Each node can be run in non-virtual mode, with one processor performing communication and

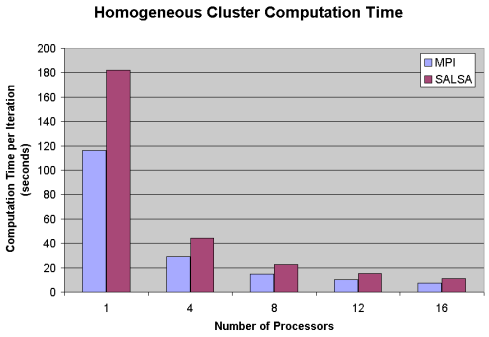

Figure 5. The time spent per iteration by the astronomy application's evaluators computing the likelihood using SALSA and MPI on the PPC cluster.

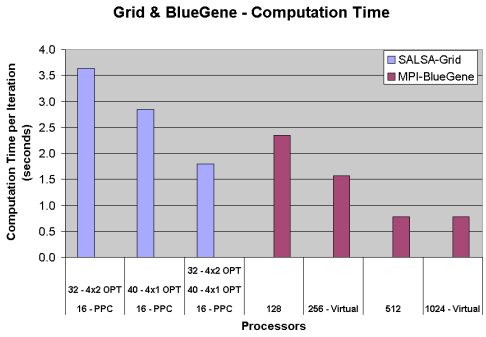

Figure 7. The time spent per iteration by the astronomy application's evaluators computing the likelihood using SALSA on the RPI Grid and MPI on the BlueGene.

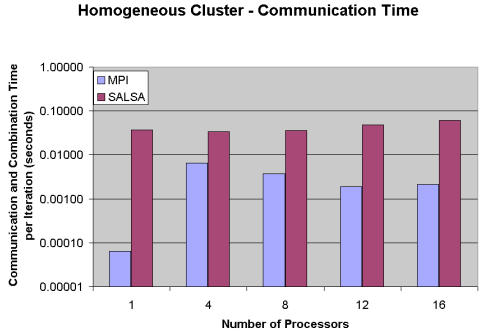

Figure 6. The time spent by the GMLE framework communicating and combining the likelihood for the astronomy application during each iteration using SALSA and MPI on the PPC cluster.

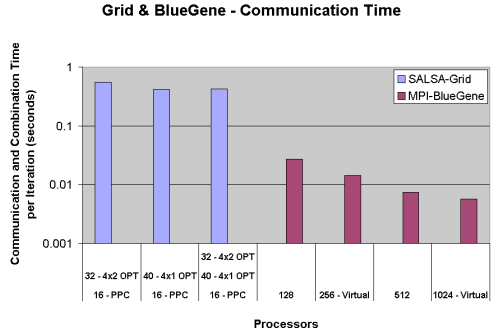

Figure 8. The time spent by the GMLE framework communicating and combining the likelihood for the astronomy application during each iteration using SALSA on the RPI Grid and MPI on the BlueGene.

the other computation, or in virtual mode, with both processors performing computation and communication. The current system consists of 5 partitions, one 512 node partition, and four 128 node partitions.

6.2 Cluster Performance

The SALSA and MPI implementations of the astronomy application are compared on a homogeneous cluster. Figure 5 shows the average computation time used per iteration with varying numbers of the PPC cluster's processors. Figure 6 shows the average time spent by the GMLE package per iteration performing communication and combining the evaluation results. These results show that using SALSA/Java is 1.5 times slower than MPI/C for computation, and 17 times slower for communication and combination of results. In general, the communication and combination time is very small compared to the computation time, so the main concern is the efficiency of the implementation of the likelihood evaluation.

6.3 Grid Performance

While MPI outperformed SALSA on a homogeneous cluster, it is not easy to run MPI on a heterogeneous cluster, which requires additional libraries such as Globus [11] and MPICH-G [12]. It also requires multiple binaries to be built for different architectures and operating systems. SALSA, on the other hand, uses the Java Virtual Machine (JVM) which allows for transparent execution across heterogeneous architectures, and can easily be used in a grid environment. Additionally, as shown in other work [25, 14], developers using the SALSA implementation can utilize the dynamic reconfiguration capabilities of IOS without code

modification.

Figure 7 shows that the computation time per iteration of the astronomy application on the PPC cluster and 4x1 OPT cluster, the PPC cluster and the 4x2 OPT cluster, and all three clusters. Using all three clusters, a 4 times speedup was gained by using SALSA on a grid like environment, compared to MPI on the cluster environment and a 65 times speedup over using a single PPC processor. Figure 8 shows the communication and combination overhead of SALSA scales well on the grid environment. Communication and combination overhead using the grid increased compared to the homogeneous cluster because of communication being done over RPI's wide area network, which is significantly slower than the inter-processor communication on the homogeneous cluster.

6.4 BlueGene Performance

Figure 7 shows the computation time for each iteration of the astronomy application using 128 and 512 node partitions of the BlueGene in virtual and non-virtual mode. Using the BlueGene in virtual mode resulted in the best performance, because the computation to communication ratio is still computation heavy. Using 128 nodes in virtual mode resulted in a 5 times speedup over MPI on the homogeneous cluster and 74 times speedup over a single PPC processor, and using 512 nodes in virtual mode resulted in a 9.5 times speedup over MPI on the homogeneous cluster and 148 times speedup over a single PPC processor. Figure 8 shows the communication and combination overhead of using MPI on the BlueGene. This overhead is small and scales well, however increasing from 512 to 1024 processors shows that the application has reached the limits of the scalability of the sample data set. A larger or more divisible data set would result in greater scalability as communication and combination overhead seem to be comparatively low.

7 Related Work

In addition to the astronomy and particle physics applications discussed in this paper, MLE is being used in a wide range of scientific applications. For example, MLE is used by Efstathiou to analyze low cosmic microwave background radiation [9]. In environmental science, Cohn uses MLE to estimate the amount of contaminants in rivers [7]. Stamatakis et al. use MLE to compute large phylogenetic trees based on taxonomic data [23]. MLE is also being used to determine genetic disease risk from individual genotype data [5], to analyze epidemiological data [21], and to perform genetic studies of allelic dropout rates [15].

Other research efforts have focused on providing grid-enabling frameworks for scientific applications. The Cactus system [2] targets numerical relativity. Cactus allows users to write application-specific modules which interface with the Cactus framework to run on a grid. NetSolve [6] provides a client-agent-server architecture to solve problems in computational science. Our framework differs from these in its focus is on maximum likelihood, and provides a generic interface to a framework which uses multiple search methods and can execute on a wide range of computing environments. Nimrod/0 [18] is a tool which enables parameter estimation on grids and provides a web interface. This work differs from Nimrod in that researchers are not only able to implement scientific models, but different search methods as well all though a simple API and the available execution environments are not limited to grids.

8 Conclusions

The GMLE framework allows for easily interchangeable scientific functions, distributed evaluation strategies and maximum likelihood search methods. The framework leaves many avenues open for future work. One possible execution environment, which is becoming increasingly effective for large scale scientific computing is the Internet. Applications such as SETI@home [3] and Folding@home [17] use the BOINC [4] computing framework to distribute computation across large numbers of volunteered computing resources. Work is currently being done to expand the GMLE framework to work with BOINC providing maximum likelihood evaluation over the Internet. Additionally, most maximum likelihood search methods that have been developed are iterative and may not scale well to a distributed setting. The GMLE framework is allowing for research into new search methods that may work better in modern distributed computing environments.

Rensselaer's Computational Center for Nanotechnology Innovation (CCNI) hosts a set of computing clusters including a 16K-node BlueGene supercomputer with 32K processors. Examining how maximum likelihood search can scale to such massive numbers of processors will prove to be an interesting area of research, that will require new concurrent searching methods and strategies for data distribution and distributed likelihood evaluation.

GMLE provides an easy to use package for scientific researchers in various disciplines who stand to gain significant increases in the performance of their maximum likelihood computations by utilizing different distributed computing environments. This paper presents the framework of GMLE, the currently implemented search methods, and gives examples of how to use this software. Examples of the speedup that can be gained by using GMLE on an IBM BlueGene supercomputer, a homogeneous computing cluster, and a heterogeneous grid are given. The GMLE framework is shown to be scalable to over 1,000 processors on

the BlueGene. As the amount of data researchers gather increases at a much higher rate than processing power, distributed computing becomes a very important component in making data analyses feasible. The GMLE package provides an easy to use tool for scientific researchers looking to harness the power of modern distributed computing environments, in addition to aiding research into concurrent maximum likelihood search methods and efficient distributed computing.

References

[1] J. e. a. Adelman-McCarthy. The 6th Sloan Digital Sky Survey Data Release, http://www.sdss.org/dr6/, July 2007. ApJS, in press, arXiv/0707.3413.

[2] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Supercomputing 2001 (SC 2001)*, Denver, November 2001.

[3] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[4] D. P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *e-Science*, pages 196–203. IEEE Computer Society, 2005.

[5] T. Becker and M. Knapp. Maximum-likelihood estimation of haplotype frequencies in nuclear families. *Genetic Epidemiology*, 27:21–32, May 2004.

[6] H. Casanova and J. Dongarra. NetSolve: A network-enabled server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, Fall 1997.

[7] T. A. Cohn. Estimating contaminant loads in rivers: An application of adjusted maximum likelihood to type 1 censored data. *Water Resources Research*, 41, July 2005.

[8] H. A. Cramer. *Mathematical Methods of Statistics*. Princeton University Press, 1958.

[9] G. Efstathiou. A maximum likelihood analysis of the low cosmic microwave background multipoles from the wilkinson microwave anisotropy probe. *Monthly Notices of the Royal Astronomical Society*, 348:885–896, March 2004.

[10] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley and Sons Ltd, Chichester, England, 1998.

[11] I. Foster and C. Kesselman. Globus: A toolkit-based grid architecture. In *The Grid: Blueprint for a New Computing Infrastructure*, pages 259–278. Morgan Kaufmann, 1999.

[12] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A grid-enabled implementation of the Message Passing Interface. *J. Parallel Distrib. Comput.*, 63(5):551 – 563, 2003.

[13] K. E. Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela. The internet operating system: Middleware for adaptive distributed computing. *International Journal of High Performance Computing Applications (IJHPCA), Special Issue on Scheduling Techniques for Large-Scale Distributed Platforms*, 10(4):467–480, 2006.

[14] K. E. Maghraoui, T. J. Desell, B. K. Szymanski, and C. A. Varela. Dynamic malleability in mpi applications. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, pages 591–598. IEEE Computer Society, May 2007.

[15] C. R. Miller, P. Joyce, and L. P. Waits. Assessing allelic dropout and genotype reliability using maximum likelihood. *Genetics*, 160:356–366, January 2002.

[16] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[17] V. Pande et al. Atomistic protein folding simulations on the submillisecond timescale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2002. Peter Kollman Memorial Issue.

[18] T. Peachey, D. Abramson, and A. Lewis. Model optimization and parameter estimation with nimrod/o. In *International Conference on Computational Science*, University of Reading, UK, May 2006.

[19] J. Purnell, M. Magdon-Ismail, and H. Newberg. A probabilistic approach to finding geometric objects in spatial datasets of the Milky Way. In *Proceedings of the 15th International Symposium on Methodoligies for Intelligent Systems (ISMIS 2005)*, pages 475–484, Saratoga Springs, NY, USA, May 2005. Springer.

[20] C. Reina, P. Bradley, and U. Fayyad. Clustering very large databases using mixture models. In *Proc. 15th International Conference on Pattern Recognition*, 2000.

[21] S. Selvin. *Statistical Analysis of Epidemiologic Data*. Oxford University Press, 2004.

[22] J. A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Publishing, 2005.

[23] A. Stamatakis, T. Ludwig, and H. Meier. Raxml-iii: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, December 2004.

[24] C. Varela and G. Agha. Programming dynamically reconfigurable open systems with SALSA. *ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings*, 36(12):20–34, Dec. 2001. http://www.cs.rpi.edu/~cvarela/oopsla2001.pdf.

[25] W. Wang, K. E. Maghraoui, J. Cummings, J. Napolitano, B. Szymanski, and C. Varela. A middleware framework for maximum likelihood evaluation over dynamic grids. In *Second IEEE International Conference on e-Science and Grid Computing*, page 8 pp, Amsterdam, Netherlands, December 2006.