Procedia Computer Science

Volume 29, 2014, Pages 256–269

ICCS 2014. 14th International Conference on Computational Science

Performance Improvements for a Large-Scale Geological Simulation

David Apostal[14], Kyle Foerster[24], Travis Desell[14], and William Gosnold[34]

[1] Department of Computer Science
[2] Department of Electrical Engineering
[3] Harold Hamm School of Geology and Geological Engineering
[4] University of North Dakota, Grand Forks, ND 58203, USA

Abstract

Geological models have been successfully used to identify and study geothermal energy resources. Many computer simulations based on these models are data-intensive applications. Large-scale geological simulations require high performance computing (HPC) techniques to run within reasonable time constraints and performance levels. One research area that can benefit greatly from HPC techniques is the modeling of heat flow beneath the Earth's surface. This paper describes the application of HPC techniques to increase the scale of research with a well-established geological model. Recently, a serial C++ application based on this geological model was ported to a parallel HPC applications using MPI. An area of focus was to increase the performance of the MPI version to enable state or regional scale simulations using large numbers of processors. First, synchronous communications among MPI processes was replaced by overlapping communication and computation (asynchronous communication). Asynchronous communication improved performance over synchronous communications by averages of 28% using 56 cores in one environment and 46% using 56 cores in another. Second, an approach for load balancing involving repartitioning the data at the start of the program resulted in runtime performance improvements of 32% using 48 cores in the first environment and 14% using 24 cores in the second when compared to the asynchronous version. An additional feature, modeling of erosion, was also added to the MPI code base. The performance improvement techniques under erosion were less effective.

Keywords:

Contents

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.023

1 Introduction

Current physical technology is incapable of probing physical phenomena to sufficient depths in the Earth's crust to accurately develop geological models, especially for gravitational, magnetic, electrical and thermal potential fields. The challenge with both forward and inverse modeling of unknown sources for potential fields is that a large number of source configurations can yield nearly identical results. One dimensional and 2-D models in particular can yield invalid results even with reasonable assumptions on key parameters. Models of the Earth's internal heat can be further complicated by transient signals due to ground water convection on local and regional scales, magma intrusion, crystallization and cooling at different depths, variation in the distribution of radioactive heat sources and climate driven changes in ground surface temperature. One of the most useful tools available to the geological community is the adoption of an adequate simulation that can process and evaluate the efficacy of geothermal systems, with particular concern being simulations at a state or regional scale.

There are numerous simulation models used in geothermal studies. This work extends a well-established simulation model that has been used at the University of North Dakota [5, 7, 14, 8]. This model performs a finite-difference heat flow simulation with convection and conduction. The model can be used to study a 2-D or 3-D geologic region. Several characteristics of the materials in the region are considered when measuring heat flow. The simulator that implements this model divides the region into cells and uses the current temperature of each cell to calculate the temperatures of each cell at the next timestep. Calculations for conduction require more computations than the calculations for convection. A Fortran implementation was recently ported to C++ to allow for easier maintenance and improvement of the program. While converting the program to C++, a number of enhancements were made to the application and the geological model. Some of the enhancements include:

- increasing the number of allowed material properties from eight to up to 99,
- incorporating OpenGL for a better visual representation of the simulation and the introduction of 3-D visualization,
- and expanding the geological model from two dimensions to three dimensions which allows for simulations with greater detail of larger areas.

With the intent of running simulations on a state or regional scale, multiple versions of the program were implemented to take advantage of HPC environments such as MPI [16]. A

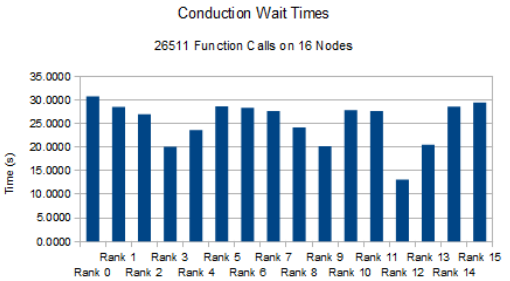

Figure 1: Imbalance caused by conduction operations.

synchronous MPI version of the program was implemented and tested on a HPC cluster using 28 Sun Microsystems x2200 compute nodes. The synchronous MPI version demonstrated 24x speedup over the serial C++ version of the program.

However, there was a problem with the synchronous communication, where the processes are blocked from performing any computations when sending data to or receiving data from another process. In the geology simulation conduction and convection operations require that results for each cell be communicated with adjacent cells at the end of each time step. With multiple conduction and convection loops per time step, this can result in large communications latencies over the course of a simulation, particularly when the sending and receiving processes are running on separate physical processors.

To reduce latencies associated with synchronous communication, the MPI implementation was modified to use asynchronous communication. Asynchronous communication enables greater overlapping of communication and computation. Asynchronous communication improved performance by an average of upto 46% using 48 cores over synchronous communication.

Another source of latency occurs when the work load associated with conduction and convection processing is not evenly distributed among processes. Processes with less work have to wait for other processes. An analysis of the asynchronous simulation application using an input file with a mostly balanced load of conduction and convection operations found differences in wait times for both conduction and convection. The amount of time a process is waiting corresponds to the time another process takes to perform calculations and return the results of those calculations. The longest wait times were 2.35x and 2.9x longer than the shortest wait times for conduction and convection operations respectively. The variations in wait times across processes for conduction is shown in Figure 1. Figure 2 shows the imbalance across processes for convection.

Repartitioning the data at the start of the program, or semi-static load balancing, was also used to improve the geology simulation. Two similar approaches were implemented. These approaches are based on the distribution of conduction and convection operations throughout the region of study. A scoring system for evaluating each approach helps to select how the data will be divided among the processes. Average gains of 26% using 48 cores and 14% using 56 cores in separate environments were observed. The repartitioning approaches should only be applied in cases where the number of conduction and convection operations performed by each process is consistent throughout the simulation.

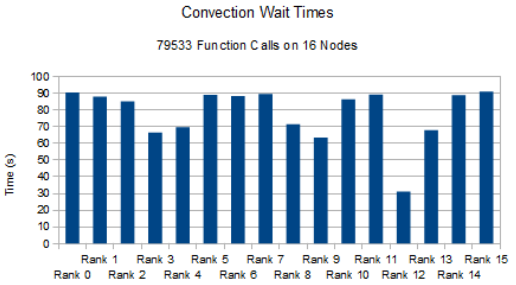

Figure 2: Convection is uneven across processes.

2 Related Work

Computer applications can achieve reductions in execution times by dividing the program in ways so that different tasks execute simultaneously on different processors, by dividing the data across multiple processors, or some combination of both task and data division. This section describes several approaches to improving performance with load balancing. *Static* load balancing assigns a distribution of work to each process. *Semi-static* load balancing estimates the cost of different data distributions at program initialization and attempts to select a distribution that will perform well. The repartitioning of data described in this paper falls into this category. *Dynamic* load balancing monitors system performance and redistributes the load during runtime.

There have been several studies involving approaches for partitioning large graphs. Leland *et al.* showed that static load balancing algorithms can be effective when applied appropriately to the right problem and computing environment [13]. For example, a partition method that causes a large amount of communication impacts performance less in a computation-bound problem.

Initial partitioning of large data models for power or water system equipment has been studied by Capko *et al.* The authors identified an algorithm for initial partitioning when the network graph is disjoint [4]. The geology simulator does not process disjoint graphs, but the grid representing the region being simulated can easily exceed millions of cells.

Meyer *et al.* developed a method for measuring the computational cost associated with read accesses needed to render images of geological data. With the authors' approach, system load is measured as a single integral of the cost function rather than a sum of a series [15]. They describe their approach to semi-static and dynamic methods of load balancing.

PREMA is a system that enables dynamic load balancing in the context of applications that are asynchronous and highly adaptive. It monitors the performance of physical processors. PREMA uses mobile objects to move tasks and data to another processor when load imbalances are detected [2]. PREMA has been shown to perform well compared to other dynamic load balancing systems "in terms of minimizing idle cycles due to workload imbalances [3]". The wait times in Figures 1 and 2 of the geology simulator correspond to the idle cycles in PREMA. The geology simulator is not asynchronous but is, when modeling erosion, highly adaptive because erosion effectively eliminates conduction and convection operations.

A different approach to dynamic load balancing is Adaptive MPI. Where PREMA is a system on which one can use different MPI implementations, Adaptive MPI is a customized implementation of MPI [9]. Adaptive MPI provides a familiar programming environment and

```
void conduction() {
    for (int k = 0; k < num_slices; k++) {
        for(int i = st_index; i < en_index; i++) {
            for(int j = 0; j < num_cols; j++) {
                conduction_helper(i,j,k);
            }
        }
    }
    update_temp_mpi();
    swap_temp_array();
}
```

Figure 3: Synchronous Communication for Conduction

integrates with a load balancing framework called Charm++ [12]. Like PREMA, Adaptive MPI/Charm++ use mobile objects to migrate MPI threads from one processor to another. Both also can use multiple virtual MPI processes per physical processor in order to maximize performance by task switching when a process is idle during communication.

3 Approach

3.1 Asynchronous MPI Implementation

The initial MPI implementation used synchronous communications as shown in Figure 3. To improve the MPI version overlapping of communication and computation using asynchronous communication was implemented by dividing the data cells into two areas: edge cells and non-edge cells. The results of computations for edge cells must be communicated to another process at the end of each simulation time step. The overlapping of communication and computation is achieved by first calculating and updating the edge cells and then sending them with asynchronous MPI send and receive functions. While a process is waiting for the asynchronous communication to finish, it finishes performing calculations on all non-edge cells. Once the process finishes updating all of the non-edge cells, it makes sure the send and receive operations have completed by calling the MPI_Waitall function to verify that the asynchronous communication has finished. An example of overlapping the communication and computation of the program is shown in Figure 4. This process demonstrates the greatest degree of performance when the computation times are greater than or equal to communication times. This allows for the largest amount of overlapping of communication and computation and produces the largest performance gain when compared to the synchronous version of the program.

3.2 Initial Data Partitioning

The MPI-based simulator divided rows of data evenly among the available processes. Each process performed computations on a contiguous block of the input file. Each block had approximately the same number of rows. For example, if the simulator processed an input file with 80 rows of data on a cluster with four processors, each processor processed a block of 20 contiguous rows of data.

However, this approach does not consider the number of conduction and convection operations indicated by individual rows in the input file.

Consider the example distribution of conduction and convection operations shown in Table 1. The table shows the total combined number of conduction and convection operations in each

```
void conduction() {
    for (int k = 0; k < num_slices; k++) {
        for(int j = 0; j < num_cols; j++) {
            conduction_helper(st_index,j,k);
            conduction_helper(en_index-1,j,k);
        }
    }
    update_temp_mpi();
    for (int k = 0; k < num_slices; k++) {
        for(int i = st_index+1; i < en_index-1; i++) {
            for(int j = 0; j < num_cols; j++) {
                conduction_helper(i,j,k);
            }
        }
    }
    if(my_rank == 0 || my_rank == comm_sz - 1) {
        MPI_Waitall(num_cols,request1,MPI_STATUS_IGNORE);
        MPI_Waitall(num_cols,request2,MPI_STATUS_IGNORE);
    }
    else {
        MPI_Waitall(num_cols,request1,MPI_STATUS_IGNORE);
        MPI_Waitall(num_cols,request2,MPI_STATUS_IGNORE);
        MPI_Waitall(num_cols,request3,MPI_STATUS_IGNORE);
        MPI_Waitall(num_cols,request4,MPI_STATUS_IGNORE);
    }
    swap_temp_array();
}
```

Figure 4: Asynchronous Communication for Conduction

row of an input file. With this input and the original approach to work load distribution, the load would be divided among four processes as shown in Table 2. The rank 1 process has nearly four times more load than the rank 0 process. More analysis of the simulator input files is needed in order to create a more even division of load.

Table 1: Example Load Distribution in an Input File

Load	1	2	4	9	9	9	6	7	4	4	2	1
Row	0	1	2	3	4	5	6	7	8	9	10	11

Table 2: Load Balance

Load	7	27	17	7
Rank	0	1	2	3

3.2.1 Constraints

The geology simulation application has a number of constraints with respect to how the data is divided and distributed. Informally, each block of data must be contiguous and not overlap with any other block. Also, each process must perform calculations on its associated data.

More formally, the constraints are as follows.

- Block 0 is comprised of data from rows 0 through (*block 0 size - 1*). Block n is comprised of data from rows (*block n-1 size*) through (*block n size - 1*). Block n+1 is comprised of data from (*block n size*) through *the number of rows - 1*).

- Process rank *0* must process block *0*. Process rank *n* must process block *n*. Process rank *n + 1* must process block *n + 1*.

3.2.2 Load Distribution Approaches

The first step in creating a more balanced load distribution is counting the number of conduction operations and convection operations indicated in the input file. This requires reading each code in each row for each slice.

An optimal load per process must be determined before attempting to balance the load. This is found by dividing the total number of convection and conduction operations for all input file rows by the number of processes available. From the example above, a load of approximately 14.5 conduction and convection operations per process would be a near optimal distribution of work. We assume the simulator runs on a cluster with a homogeneous processing environment. If the environment is heterogeneous, another approach for identifying the optimal load per process must be determined.

A greedy approach to load balancing involves assigning rows of data to a process until the total load for the process exceeds the optimal load per process calculated earlier. When the load assigned to a process exceeds the average load, that process is deemed fully loaded, and the next row of data shall be assigned to the next process. In this approach the last process will have a smaller load than the others unless the load in each block is equal. Table 3 shows the load distribution using this approach for four processes and an optimal load value of 14.5. We call this a top-down greedy approach because the loads are accumulated by reading from the top of the input file.

Table 3: Top-Down Greedy Load Balance

Load	16	18	17	7
Rank	0	1	2	3

A similar greedy approach using the same optimal load value involves assigning rows starting at the bottom of the input file and starting with the last process. Table 4 shows the resulting load distribution using this bottom-up greedy approach.

Table 4: Bottom-Up Greedy Load Balance

Load	7	18	15	18
Rank	0	1	2	3

3.2.3 Scoring

At this point there are three different load distributions to consider. One must be selected in order for the simulation to proceed. The approach we use to select a load distribution is based on the cumulative differences between the optimal load and each process load. Using this approach the distribution in Table 2 has a score of 30. The two greedy approaches in Table 3 and Table 4 have scores of 15. Therefore, the distribution from the top-down greedy approach would be selected because no other approach has a lower score.

After a distribution has been selected, the input file read pointer is restored to the point saved at the beginning of the load balancing process. The details of the selected distribution are broadcast to all process ranks.

3.3 Erosion

To allow the program to simulate a larger range of subdomains within the geothermal research area, the ability to simulate erosion was added to the program. This capability allows for the simulation of environments that are prone to erosion over extended periods of time, such as canyons.

This feature was implemented by giving each cell erosion properties that define the erosion rate in a given direction in $m^3/year$ and a percent volume remaining. For each time step of the simulation, the program checks if a cell is adjacent to an air cell, which is defined as a cell with 0% volume remaining. If a cell is adjacent to an air cell, the program subtracts volume from the cell's remaining volume based on the direction of erosion and the cell's defined erosion rate for the given direction.

Along with the inclusion of erosion calculations, the conduction and convection calculations were updated to handle the addition of air cells and erosion. This was done by updating the conduction calculations to prevent the transfer of heat if one cell is an air cell and the convection calculations were also updated to prevent convection from occurring for a given cell if it is adjacent to an air cell. The convection and conduction calculations are also only performed on non-air cells.

4 Results

The geology simulator was tested with four configurations: synchronous communication with no load balancing, synchronous communications with load balancing, asynchronous communications with no load balancing, and asynchronous communications with load balancing enabled. The average runtimes of four runs are reported. Each test was measured with MVAPICH2 [10].

Also, each configuration was tested with three data files: small 2-D, small 3-D, and large 3-D. The data files had 19250.67 year time-steps and 20E year runtimes. Sometimes during load balance testing, the greedy approaches did not result in a more optimal load per process than the default approach of dividing rows of data evenly among the processes. On these occasions the default approach was used. These occasions are identified during the discussion of each input file.

4.1 Experimental Testbed

Tests were conducted on two HPC cluster environments. One cluster called Hodor is housed at the University of North Dakota Computational Research Center. Hodor has 32 nodes. Each node has two quad core Intel 2643 processors with 64GB of RAM. Hodor nodes are connected by an InfiniBand FDR link.

The second cluster is Stampede at the Texas Advanced Computing Center at the University of Texas-Austin. Stampede has 6,400 nodes. Each node has two eight core Intel E5 processors with 32GB of RAM. The Stampede nodes are also connected by an InfiniBand FDR link. Access to Stampede was made possible by the Extreme Science and Engineering Discovery Environment (ESEDE) project.

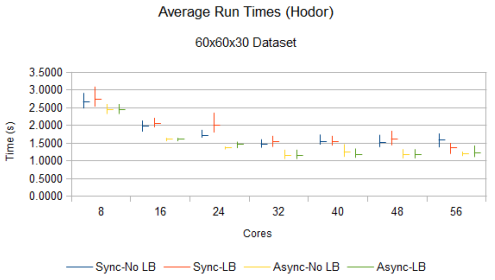

Figure 5: Results for Small 3-D input file on Hodor. The horizontal ticks are the average runtimes.

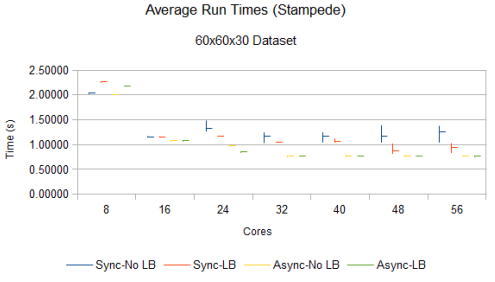

Figure 6: Results for Small 3-D input file on Stampede.

Each simulator configuration was tested with between eight and 56 cores on both HPC clusters. The job submission scripts on each cluster were set to use the maximum number of cores available before using any additional nodes. There was less communications overhead on Stampede because each processor has more cores than the processors on Hodor.

4.2 Small 3-D Results

With the Small 3-D data set the average runtime performance decreases as more cores are utilized. However, the performance gains eventually level off because the program cannot overcome the communications latency caused by larger numbers of cores. On Stampede there was a noticable variation in the synchronous runtimes with 24 or more cores. This is an effect of the number of cores per processor and the number of processors per node. there are 16 cores per node on Stampede. Using more than 16 cores increases the network overhead. However, the synchronous configurations were fairly consistent. The configurations with asynchronous communications perform consistently better than their synchronous counterparts. On occasion on both Stampede and Hodor the load balancing configurations actually perform less than without load balancing.

With this input file and with 40 cores the default approach for dividing data among the cores had the lowest load balancing score.

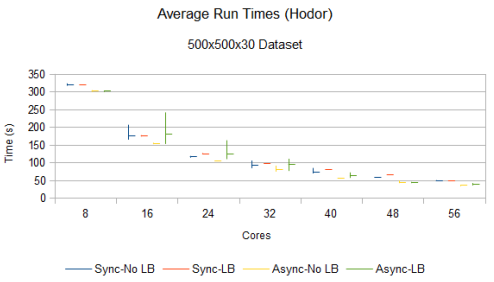

Figure 7: Results for Medium 3-D input file on Hodor.

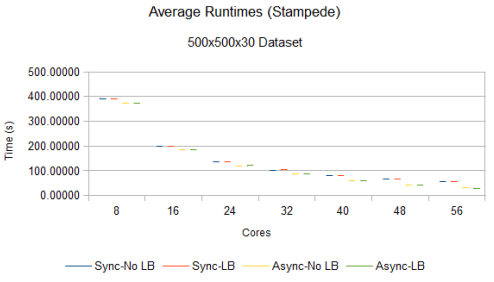

Figure 8: Results for Medium 3-D input file on Stampede.
B

4.3 Medium 3-D Results

The medium 3-D input file produced similar results as the small 3-D input file as shown in Figure 7 and Figure 8. There is less variation between minimum and maximum times with this dataset than with the small 3-D input file because the simulator spends a smaller percentage of time doing communications. The tests using eight cores used the default approach of dividing rows of data evenly among the cores instead of one of the greedy approaches because the default approach had the lowest load balancing score.

4.4 Large 3-D Results

Figure 9 and Figure 10 show again that the performance of the simulator continues to improve as the number of processors increases. There is almost no difference between the minimum and maximum times for each configuration. In the tests using 48 and 56 cores the default approach of distributing rows of data among the cores had the lowest load balancing score.

4.5 Data Partitioning Results

The top-down and bottom-up greedy approaches to data partitioning were evaluated along with the original distribution of data using different combinations of nodes and processes per node. After initial testing, two modifications to the partitioning schemes were made. First, it was observed that the time for conduction operations was longer than the time for convection operations. On the Hodor cluster the time for a conduction operation takes 4.36x longer than a

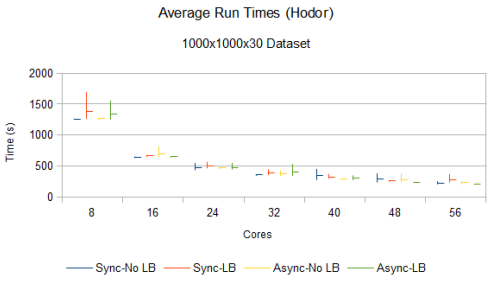

Figure 9: Results for Large 3-D input file on Hodor.

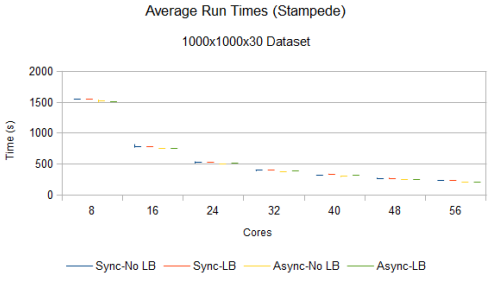

Figure 10: Results for Large 3D input file on Stampede.

convection operation. As a result a factor was added as each conduction operation was counted.

The second modification was made after it was observed that in some cases of the top-down and bottom-up approaches some processes were being assigned zero rows of data to process. In particular, this was observed with data where the load per row was fairly uniform. The greedy approach forced an entire row of data to be assigned to a process if the process load was less than the previously computed optimal load. To address this aggressiveness a look ahead was introduced when totaling the load for a process. The modification allows the top-down and bottom-up approaches to consider if the load for the current process will be closer to the optimal load with or without the next row of data. If the load is closer to the optimal load without adding the next row of data, then the process is deemed fully loaded, and the next row of data will be assigned to the next process.

These changes led to more uniform process loads as seen in Figures 11 and 12. Also, the maximum difference in wait times between processes was reduced after repartitioning the input data. Using the same mostly balanced input file, the longest wait times were reduced to 2.04x and 2.06x longer than the shortest wait times (from 2.35x and 2.9x before data repartitioning) for conduction and convection respectively.

4.6 Erosion Results

The addition of erosion to the program allows for the simulation of environments that change due to erosion over extended periods of time. One example of this is shown in Figure 13, where the simulation without erosion is shown beside a simulation with erosion. For this test a small grove of air is in the top middle of the simulation area. As the simulation ran, the area around

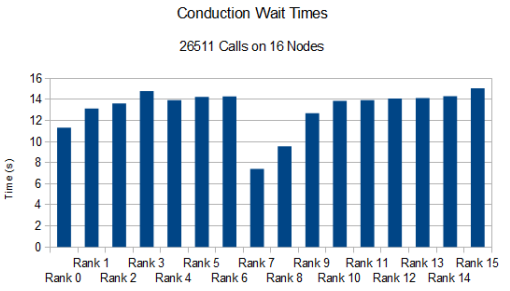

Figure 11: Times for conduction are more uniform across processes.

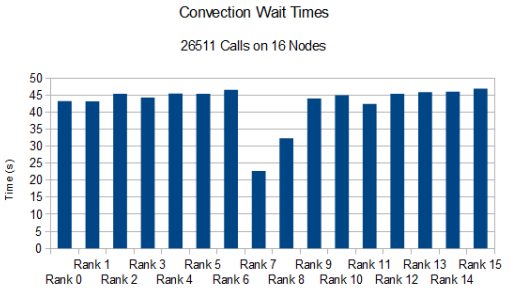

Figure 12: Convection time is more uniform after modifying the data partitioning approaches.

the air slowly eroded outward demonstrating the ability of the program to simulate areas with changing environments due to erosion.

This raises an issue concerning initial load balancing approaches. The eroded area represents areas where neither conduction nor convection occurs. However, the initial load balancing was based on conduction and convection operations occuring in the eroded areas. When simulating erosion, the observed performance improvement for initial load balancing was only 2%.

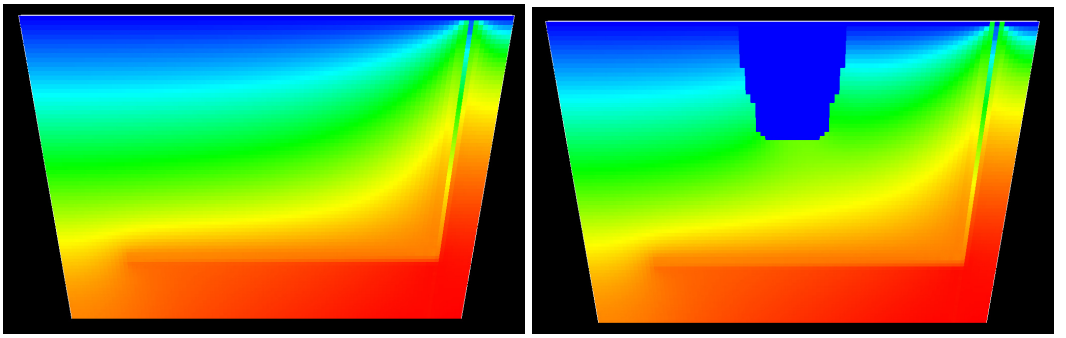

Figure 13: Simulation Without Erosion (left) and With Erosion (right)

5 Conclusion

This paper presents HPC improvements to a geological simulation application that required only modest changes to an existing MPI application. The initial MPI implementation used synchronous communications. Enhancements to the MPI implementation were made to enable state and regional scale simulations using large numbers of processors.

One technique for improving performance was using asynchronous messaging instead of synchronous messaging. Asynchronous messaging allowed more overlapping of computation and communication. Tests showed that asynchronous messaging increased runtime performance over synchronous messaging by averages of 28% using 56 cores in one environment and 46% using 56 cores in another environment.

Another technique for improving performance involved load balancing of the processes by repartitioning the initial input data. Three approaches for data partitioning were combined with a scoring system to select a distribution of data that was closest to an optimal per process distribution of load. Repartitioning of data in this manner resulted in runtime performance improvement averages of 32% using 48 cores and 14% using 24 cores in two different environments. However, the benefits of repartitioning were dependent on the data itself as well as the computing environment.

An area of future work involves implementing dynamic load balancing to improve further the performance of the simulator. This work should help when modeling erosion or when sources of heat move in a geologic region because the amount of computation done on each process changes as the simulation progresses.

Another area of interest involves applying optimizations specific to CUDA and the GPGPU environment. The geology simulator may particularly benefit from the combined use of MPI and CUDA. Clusters of GPGPU devices have been applied in wide-ranging fields [6, 11, 1].

Further enhancements to the existing data partitioning are also envisioned. For example, another set of load distributions can be found using a different optimal value or a different method of scoring. Also, more flexibility in data partitioning would be possible if individual rows of data could be partitioned instead of the current data partitioning based on the loads in complete rows of the input file.

One potential limitation to further use of the application is the size of the input file required to perform larger scale simulations. Future work designing alternative input file formats, including binary files or distributed data files, could remove this limitation.

References

[1] David Apostal, Kyle Foerster, Amrita Chatterjee, and Travis Desell. Password recovery using MPI and CUDA. In *IEEE International Conference on High Performance Computing (HiPC)*, Pune, India, 2012.

[2] K. Barker, A. Chernikov, N. Chrisochoides, and K. Pingali. A load balancing framework for adaptive and asynchronous applications. *Parallel and Distributed Systems, IEEE Transactions on*, 15(2):183–192, 2004.

[3] Kevin J. Barker and Nikos P. Chrisochoides. An evaluation of a framework for the dynamic load balancing of highly adaptive and irregular parallel applications. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, SC '03, pages 45–, New York, NY, USA, 2003. ACM.

[4] D. Capko, A. Erdeljan, M. Popovic, and G. Svenda. An optimal initial partitioning of large data model in utility management systems. *Advances in Electrical and Computer Engineering*, 11(4):41–46, 2011.

[5] S.L. de Silva and W.D. Gosnold. Episodic construction of batholiths: Insights from the spatiotemporal development of an ignimbrite flare-up. *Journal of Volcanology and Geothermal Research*, 167(1):320–335, 2007.

[6] S. Dosopoulos, J. D. Gardiner, and J.-F. Lee. An MPI/GPU parallelization of an interior penalty discontinuous galerkin time domain method for maxwells equations. *Radio Science (2011)*, 46, 2011.

[7] W.D. Gosnold. Basin-scale groundwater flow and advective heat flow: An example from the northern great plains. *Geothermics in Basin Analysis*, pages 99–116, 1999.

[8] Will Gosnold, Jacek Majorowicz, Rob Klenner, and Steve Hauck. Implications of post-glacial warming for northern hemisphere heat flow. *Transactions of the Geothermal Resources Council*, page 12, 2011.

[9] Chao Huang, Gengbin Zheng, Laxmikant Kalé, and Sameer Kumar. Performance evaluation of adaptive MPI. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '06, pages 12–21, New York, NY, USA, 2006. ACM.

[10] Wei Huang, Gopalakrishnan Santhanaraman, H-W Jin, Qi Gao, and Dhabelswar K Panda. Design of high performance MVAPICH2: MPI2 over InfiniBand. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 43–48. IEEE, 2006.

[11] D.A. Jacobsen, J.C. Thibault, and I. Senocak. An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters. *Mechanical and Biomedical Engineering Faculty Publications and Presentations*, page 5, 2010.

[12] Laxmikant V. Kale and Sanjeev Krishnan. Charm++: a portable concurrent object oriented system based on c++. *SIGPLAN Not.*, 28(10):91–108, October 1993.

[13] R. Leland and B. Hendrickson. An empirical study of static load balancing algorithms. In *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, pages 682–685, 1994.

[14] Jacek Majorowicz, Will Gosnold, Allan Gray, Jan Safanda, Rob Klenner, and Martyn Unsworth. Implications of post-glacial warming for northern alberta heat flow - correcting for the underestimate of the geothermal potential. *Transactions of the Geothermal Resources Council*, page 17, 2012.

[15] JanC. Meyer and AnneC. Elster. A load balancing strategy for computations on large, read-only data sets. In Bo Kgstrm, Erik Elmroth, Jack Dongarra, and Jerzy Waniewski, editors, *Applied Parallel Computing. State of the Art in Scientific Computing*, volume 4699 of *Lecture Notes in Computer Science*, pages 198–207. Springer Berlin Heidelberg, 2007.

[16] ANL Mathematics and Computer Science. The message passing interface. available at: `http://www.mcs.anl.gov/research/projects/mpi/index.htm(Mar,2014)`.