Predicting MPI-Based Parallel Application Performance on Workstation Clusters Using LogGP

Project Report for CS747: Advanced Computer Systems Modeling
Author: Michael J. Brim

I. Introduction

As clusters of commodity workstations continue to grow in popularity, so does their use as high-performance parallel application architectures. Current high-performance computing (HPC) clusters have processor counts in the hundreds, and many initiatives are already underway that plan to incorporate the use of clusters with thousands to tens of thousands of processors. Furthermore, much research is ongoing in the area of computational grids, where distributed HPC sites can be used together in order to solve ever-increasing size problems. In planning these large-scale systems, most researchers have been focusing on the hardware and software infrastructure requirements, while projections of application performance on these systems are few. The goal of this project is to develop a model of a common HPC parallel benchmark application that will enable accurate predictions of the performance of similar applications on these large-scale systems.

II. Parallel Application Modeling

One of the largest obstacles in producing an accurate yet generic model of application performance is determining which parts of the application and system are critical to the model's accuracy. If one tries to model too much of a particular sample application, the model will not be general enough to apply to other applications. On the other hand, making the model too general could result in a lack of meaningful observations. In an attempt to overcome these extremes, Culler et. al. [1] have created a parallel machine model called LogP that captures the essence of the important criteria in modeling parallel applications. The characteristics of parallel applications that LogP seeks to represent include computational bandwidth, communication bandwidth and delay, and the effects of different methods of distributing communication and computation within an application. One noted deficiency in LogP is that it only pertains to applications that communicate using short messages. In response to that deficiency, the model was further enhanced by Alexandrov et. al. [2] in order to allow for longer messages. This updated model, LogGP, incorporates the additional characteristic of communication bandwidth for long messages. Accounting for long messages becomes important for parallel architectures such as the IBM SP that provide increased bandwidth for larger messages.

III. Approach

The approach taken in this project is identical to that used by Sundaram-Stukel and Vernon in [3], wherein an analysis using LogGP was performed for predicting the performance of the Sweep3D ASCI benchmark on the IBM SP/2. Sweep3D is a three-dimensional particle transport application that uses MPI for communication. In [3], the

authors first develop models of the basic MPI send and receive operations as implemented in IBM's MPI. Next, they use simple communication benchmarks to derive the LogGP parameters that are used as inputs to the application model. The accuracy of the application model is then verified for processor counts up to 128. Finally, the model is used to predict the application's performance for varying problem sizes and much larger (up to tens of thousands) processor counts.

In this project, the first step is to once again develop simple models for MPI send and receive, this time using the LAM/MPI [4] implementation as the reference. LAM/MPI is a freely available implementation of MPI currently being developed at Indiana University. The reasons for choosing LAM/MPI as the reference implementation include the freely available source code and the added insight that can be gained from its authors, with whom I am personally acquainted. The models are then verified against performance measures obtained using the simple communication benchmarks from [3].

Once the accuracy of the MPI model is sufficient, the values for the LogGP parameters of the application model can be derived. The application to be modeled in this study is the LU application benchmark included with the NAS Parallel Benchmark 2.0 Suite [5]. The LU application is a simulated computational fluid dynamics (CFD) code that solves a block lower triangular-block upper triangular system of equations produced by an unfactored implicit finite-difference discretization of the Navier-Stokes equations in three dimensions. LU is similar to Sweep3D in that it uses diagonal pipelining method, also known as a "wavefront" method, to perform communication of partition boundaries. As a result of this similarity, the resulting LogGP application model is expected to be very similar to the one produced in [3].

The model will then be used to predict application performance for small processor counts from four to thirty-two. The accuracy of the model will then be verified against actual application performance as shown when running the LU application on the departmental Linux cluster. If the model proves to be accurate, the final step will be to predict the performance of LU, and by generality any similar wavefront type application, on much larger processor counts.

V. Modeling MPI Communication

i) Analysis of LAM/MPI

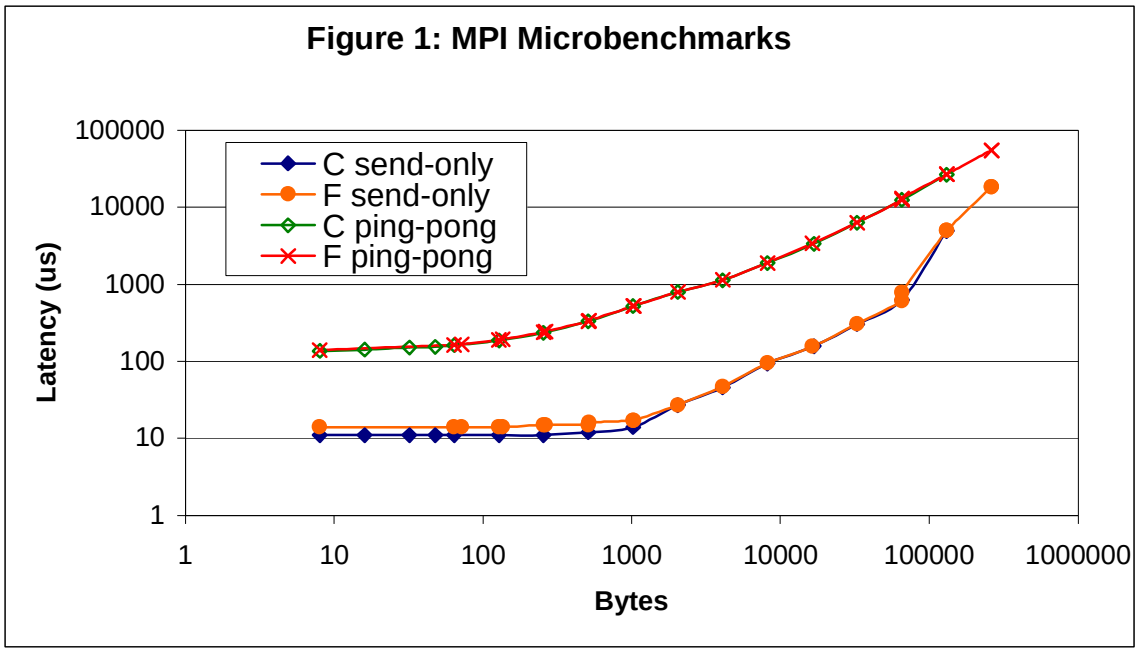
Before running the microbenchmarks to document actual MPI send/recv performance, the LAM/MPI code was first analyzed to determine the mechanisms used for sending and receiving messages of various sizes. For each message sent, LAM/MPI has the notion of an envelope, which is a specialized header that contains all the information necessary for distinguishing between messages received at some destination. In addition, LAM/MPI distinguishes between message sizes by having two categories of messages, short and long. LAM/MPI defines a short message to be any message less than or equal to 65536 bytes in length. For short messages, a call to `MPI_Send` will result in the sending of the envelope and the corresponding message all at once. For long messages, the envelope is

sent to the destination rank first, and only after the destination has acknowledged the envelope will the source then send the message contents. Calls to `MPI_Recv` result in polling the local envelope buffer to determine if the correct message envelope has been received, and then receiving the message once the envelope has been posted. For long messages, the receiving rank first sends an acknowledgment of the envelope to the source rank before beginning to receive the message contents.

ii) Benchmarking MPI Communication

In order to benchmark the performance of the LAM/MPI implementation, the same communication microbenchmarks used in [3] were obtained. The microbenchmarks, implemented in both C and Fortran, provide performance results for two types of communication, send only and ping-pong. In the send only tests, various sized blocks of data are sent from one rank using `MPI_Send` to another who receives the data using `MPI_Recv`. In the ping-pong tests, various block sizes are once again sent and received, but the receiver additionally sends the data back to the sender. Both tests report total latency for each block size. The results from running the MPI microbenchmarks are shown in **Figure 1**. For both the send only and full ping-pong benchmarks, the C and Fortran benchmarks perform similarly. The interesting observation to be made from the send plots is that there are three sections with respect to the underlying message size. The first section of the plots corresponds to message sizes that can be transmitted in a single Ethernet packet (~1500bytes) with the additional envelope header as well as TCP and IP headers, and thus messages within the range of 0 to 1400B incur almost constant latency. The next section of the send plots corresponds to messages between 1400B and 64KB. Thus this section of the plots corresponds to short messages that require multiple packets to be sent. The final section corresponds to long messages. In the case of the ping-pong plots, there is much less distinction between the various message sizes.

Figure 1: MPI Microbenchmarks



iii) MPI Communication Model

Given the results from the microbenchmarks, the following model of MPI communication was developed. Due to the one-packet phenomena, two separate values are used for the processing overhead (o) to denote whether the message is smaller (o_s) or larger (o_l) than 1400B. Similarly, the Gap per byte (G) parameter is modeled separately for each of the three message size sections (<1400B, 1400B-64KB, >64KB) discussed previously. The values for the G parameters are derived from the slopes of the curves shown in the above figure.

For messages smaller than 64KB, the envelope is sent at the same time as the message contents. Thus, the total communication can be modeled as the following, where o & G depend upon whether the message is smaller or larger than 1KB:

```
Total Comm = o + (message size * G) + L + o
```

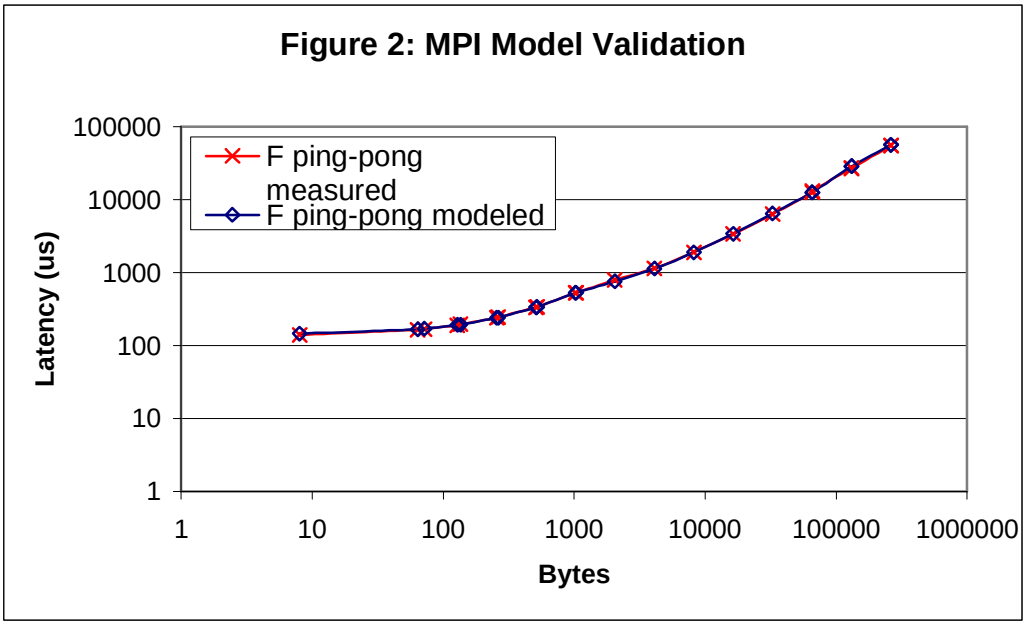
For message larger than 64KB, an extra round trip needs to be modeled to account for the sending of the envelope separate from the message contents and the extra acknowledgment from the receiver. Thus, the total communication can be modeled as:

$$\text{Total Comm} = o_s + L + o_s + o_s + L + o_l + (\text{message size} * G_c) + L + o_l$$

The following tables show the values for the model parameters for Fortran and C obtained from solving the three equations for total communication for messages less than 1400B (G_a), between 1400B and 64KB (G_b), and greater than 64KB (G_c). Rather than deriving the latency from the equations, the measured value of 60 microseconds was used to help derive the values for o_s and o_l. The accuracy of the model is shown in **Figure 2**.

Fortran Parameters	
G_a	0.372917
G_b	0.183877
G_c	0.212738
o_s	41.26
o_l	159.84
L	60

C Parameters	
G_a	0.374008
G_b	0.182775
G_c	0.216644
o_s	39.03
o_l	163.43
L	60



VI. Modeling the NAS LU Application Benchmark

i) Analysis of LU Application

The operation of the LU application can be described as having three distinct phases. The first phase corresponds to all the work needed for initializing the application, which includes the times for setting up the MPI environment, partitioning the three-dimensional grid across a two-dimensional grid of processors, and initializing various data structures. The second phase is when the real work of the application is performed, and consists of a predetermined number of symmetric successive over-relaxation (SSOR) iterations in which sweeps are made from one corner to another on the z-plane. During each iteration, there are two sweeps. The first is a northeast to southwest sweep to compute the block lower triangular solution, and the second is a southwest to northeast sweep to compute the block upper triangular solution. Both sweeps communicate using a “wavefront” pattern The wavefront communication pattern is a result of the slicing of the z-dimension in order to help pipeline the computation. After each processor has computed its portion of the z-dimension slice and communicated the boundary values to its neighbors, it can begin computation of the next slice while other processors are still working on the previous one. The final phase is used to verify the results obtained in the second phase versus the accepted values. Since the time spent in the first and third phases is insignificant in comparison to the time spent in SSOR iterations, the model of LU developed herein focuses strictly on the second phase.

ii) LogGP Model Construction

Using the results from the analysis described in the previous section, the following model of the LU application was constructed.

$$(1)\ W_{i,j} = it * jt * W_{gridpt}$$
$$(2)\ T_{jacld} = it * jt * W_{jacld}$$
$$(3)\ T_{jacu} = it * jt * W_{jacu}$$
$$(4)\ StartP_{i,j} = \max(StartP_{i-1,j} + W_{i-1,j} + T_{send} + T_{comm},\ StartP_{i,j-1} + W_{i,j-1} + T_{comm} + T_{recv})$$
$$(5)\ T_{blts} = StartP_{n,m} + K*W_{n,m} + (K-1)*(T_{jacld} + 2*T_{recv})$$
$$(6)\ T_{buts} = StartP_{n,m} + K*W_{n,m} + (K-1)*(T_{jacu} + 2*T_{recv})$$
$$(7)\ T_{SSOR} = \#iterations * (T_{blts} + T_{buts})$$
$$(8)\ T_{LU} = T_{init} + T_{SSOR} + T_{verify}$$

Equation 1 represents the time required be a single processor to compute an it * jt * 1 portion of the grid, corresponding to the work performed during one slice of the lower or upper block triangular solutions. Equations 2 and 3 correspond to the work done by a processor during one slice to compute the lower or upper triangular values of the jacobian matrix used in computing the solutions.

Equation 4 is a recursive equation used to determine the time when a processor is expected to begin its work. Since a processor cannot begin until receiving the boundary values from its north and west neighbors, the time at which it starts computation can be calculated as the maximum value of two parts. The first part is used to describe the situation when the boundary values from the west neighbor are last to arrive at $processor_{i,j}$. In this case, the time when the processor can begin is approximately equal to the starting time of its west neighbor plus the computation performed by the neighbor plus the time for the neighbor to send boundary values to its south neighbor plus the total time spent communicating boundary values with this processor. The second part is similar and corresponds to when the boundary values from the north neighbor are last to arrive. Thus, the starting time for this processor is approximately the starting time of its north neighbor plus the computation performed by the neighbor plus the time for the neighbor to send boundary values to this processor plus the time that this processor needs to receive the boundary values from the west neighbor.

Equations 5 and 6 represent the total time required for a single iteration of the block lower or upper triangular solution. The estimated total time for the lower triangular solution, T_{blts}, is shown in Equation 5. This equation is intuitive in that it represents the total time for the sweep as the starting time of the lower-right processor plus the work required by that processor for all K z-dimension slices plus the time for intermediate computation and receiving boundary values K-1 times. Since the sweep for computing the upper triangular solution is symmetric with the lower triangular sweep, the same equation is used and the appropriate values (W_{buts} and T_{jacu}) are substituted.

Equations 7 represents the total time for all SSOR iterations, corresponding to the second phase as described previously. Equation 8 is the final equation for the estimated running time for the entire application, and includes all three phases, with a single parameter for each phase. Note that the parameters for the time spent in initialization and verification of results are insignificant compared to the time spent in the SSOR iterations. As such, these

values should be obtained using measurement rather than trying to derive model equations that adequately describe their behavior.

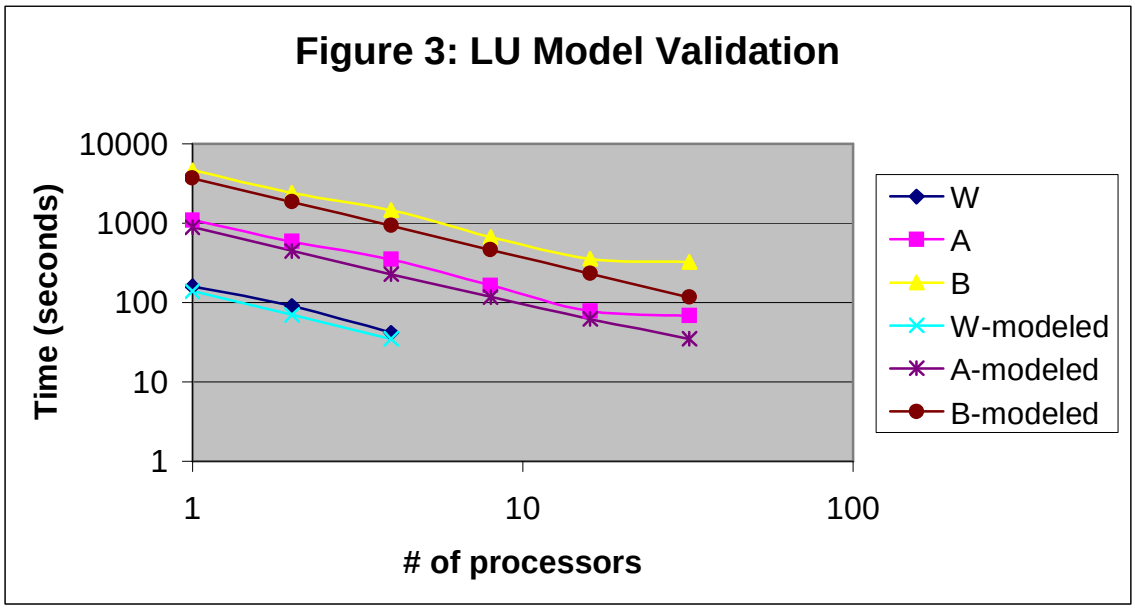
iii) Verification of LU Model

The running time of the LU application was measured using the LAM/MPI implementation of MPI on the departmental c2 cluster. Times were gathered for three different problem sizes: class W with a 33x33x33 grid, class A with a 64x64x64 grid, and class B with a 102x102x102 grid. The application was run in configurations of 1 to 32 cluster nodes. **Table 1** provides the average values of several parameters obtained from instrumentation of the application that were used as inputs to the model. Note that these values are consistent across all class and cluster sizes.

W_{blts}	3.22 µs
W_{buts}	2.85 µs
W_{jacld}	4.19 µs
W_{jacu}	4.07 µs

Table 1: Work per Grid Pt for Various Computation Phases

As mentioned previously, only the model of T_{SSOR} is validated against measured performance. **Figure 3** shows the ability of the model to predict application performance for the three class sizes and cluster sizes up to 32 nodes.



Although at first glance the model looks to be closely correlated with the measured performance, closer investigation yields notable discrepancies. For all class and cluster sizes, the model under predicts the time required for the SSOR iterations. The relative error of the model to the measured performance ranges from around 15 to 65%, which is

unacceptable for the type of modeling being performed. Furthermore, the relative error increases with the size of the cluster, limiting the model's use as a predictor of performance for larger clusters. This error for increasing cluster sizes seems to be representative of the model's inability to correctly predict the attainable speedup of the application, as shown in the figure when going from 16 cluster nodes to 32 nodes, where the actual performance shows very little speedup while the model continues to predict almost linear speedups.

Since the model is obviously insufficient in its current state, it is necessary to propose educated guesses as to the source of the discrepancies. One probable source of inaccuracy could be unrecognized synchronization effects within the communication patterns of the sweeps. Indeed, the model was modified to account for such effects using the methods in [3], but the benefits of this extension were negligible. Another possible source for the discrepancies might involve some aspect of the computation performed that is not captured in the models for T_{blts} and T_{buts}. This type of effect has already been observed as the leading cause for error in the first version of the model, where the time spent computing the jacobian matrix values (T_{jacld} and T_{jacu}) used in computation of the block triangular solutions was left out of the equations. The result of this oversight was model predictions that were more than a factor of three off from the measured performance. It is thus highly likely that computational effects may be a contributing factor to the error in the current model.

VII. Conclusions

The motivation for the work performed herein was twofold. First, there was the goal of validating the models for MPI communication developed in [3] on a different hardware platform. I believe it is clear from the results presented on the MPI models that this goal has been met. The second goal was to validate the use of LogGP to accurately predict the performance of MPI-based parallel application, specifically those that exhibit a wavefront communication pattern. The ease in deriving a new model for LU from the previous Sweep3D model described in [3] lends credence to the ability of LogGP to represent wavefront communication when combined with computational workload characteristics. However, due to the inaccuracy of the model in its present state, it would be unwise to use it to predict the performance of the LU application for greater cluster sizes. However, it is my belief that with further investigation and tuning of the model, a quite accurate version could be produced. If such a model could in fact be produced, it would serve to once again show that LogGP is a viable modeling strategy for MPI-based parallel applications.

VIII. References

[1] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation." In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[2] A. Alexandrov A., M. Ionescu, K. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model." In *Proc. 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, Santa Barbara, CA, July 1995.

[3] D. Sundaram-Stukel and M. K. Vernon, "Predictive Analysis of a Wavefront Application Using LogGP." In *Proc. 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Atlanta, GA, May 1999, pp. 141-150.

[4] LAM/MPI, http://www.lam-mpi.org.

[5] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. "The NAS Parallel Benchmarks 2.0," Technical Report NAS-95-020, December 1995.