SAND2015-8651C

Preparing for Exascale: Modeling MPI for Many-Core Systems using Fine-Grain Queues [*]

Patrick G. Bridges
Department of Computer Science
University of New Mexico
bridges@cs.unm.edu

Matthew G.F. Dosanjh
Department of Computer Science
University of New Mexico
mdosanjh@cs.unm.edu

Ryan Grant
Scalable System Software Department
Sandia National Laboratories
regrant@sandia.gov

Anthony Skjellum
Department of Computer Science and Software Engineering
Auburn University
skjellum@auburn.edu

Shane Farmer
Department of Computer Science and Software Engineering
Auburn University
shanefarmer22@gmail.com

Ron Brightwell
Scalable System Software Department
Sandia National Laboratories
rbbrigh@sandia.gov

ABSTRACT

This paper presents a fine-grain queueing model of MPI point-to-point messaging performance for use in the design and analysis of current and future large-scale computing systems. In particular, the model seeks to capture key performance behavior of MPI communication on many-core systems. We demonstrate that this model encompasses key MPI performance characteristics, such as short/long protocol and offload/onload protocol tradeoffs, and demonstrate its use in predicting the potential impact of architectural and software changes for many-core systems on communication performance. In addition, we also discuss the limitations of this model and potential directions for enhancing its fidelity.

1. INTRODUCTION

High-Performance Computing (HPC) communication system performance is critical to scientific application performance. However, recent architectural, programming model, and application changes challenge current message passing library designs, particularly those that target next-generation exascale computing systems. For example, many task application and programming model designs are stressing communication message rates at the same time that many-core processors have *reduced* the message rate that current HPC communication systems can handle [4, 3]. Many-core processors have concomitantly exposed key synchronization and threading challenges both in the MPI standard itself [20, 8] and in its implementation. Finally, network interface (NIC) design choices also present important challenges to communication library design, for example in terms of protocol offload vs. onload [10].

To address these challenges, we propose using fine-grain queueing models to analyze the key performance and resource allocation tradeoffs in HPC messaging systems. These models capture issues of resource contention, synchronization, and concurrency in modern systems that previous models do not seek to encompass, though they also have their own limitations. In addition, more complex queueing analyses can model the performance impact of complex stochastic communication workloads as opposed to simple average-case analyses, as well as more complex synchroniation behaviors. Overall, this approach will allow system designers to quantify NIC and middleware design tradeoffs more accurately without the overhead of complex simulation or prototype implementation efforts. Surprisingly, we found little if any literature on this subject.

This paper presents our basic approach to using queueing systems to model MPI point-to-point performance in a framework that encompasses the primary sources of contention and delay on modern hardware. Our contributions include:

- The description of a simple queueing model for MPI point-to-point communication that captures important performance issues such as matching cost, protocol onload/offload, and short/long message switch points in analyzing MPI bandwidth and latency;
- A validation of this model against the point-to-point performance of simple, well-known MPI benchmark running on modern MPI implementation and hardware;
- A demonstration of this model for analyzing the impact of match list length on MPI message rate on both

[*]This work was supported in part under contract Sandia National Laboratories Contract 1466715. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly-owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

many-core and heavyweight SMP systems, similar to previously published results;

- An example predictive analysis of the impact of differing numbers of MPI progress engines on message rates in next-generation many-core systems; and
- A discussion of the capabilities, challenges, and limitations on the broader use of queueing models to analyze and predict MPI communication system performance.

After providing essential background in Section 2, we describe these contributions in detail in Sections 3 though 6. Sections 7 and 8 then discuss related work and conclude, respectively.

2. BACKGROUND

Queuing models have long been used to model the performance of computing systems, including I/O, Internet and Cloud systems, as well as other complex computing setups [5, 12]. In these models, systems are represented by a set of linked queueing stations. In a closed queueing network, a fixed population of requests route between different stations, with routing potentially occurring probabilistically based on the *class* of the request. In an open queueing network, requests arrive to the system periodically, and later leave the system. In both cases, the amount of time a request takes to cycle through the system, the average utilization and residence time at each station, and the distribution of queue lengths at each station are all important metrics.

The stations themselves can take on a wide range of forms, but generally are comprised of a queue at which requests arrive according to some arrival distribution and a server which processes elements from that queue with some service distribution in an order determined by a queueing discipline. So-called M/M/1 FIFO stations are one of the most commonly used types of stations in queueing networks, and represent a station with 1 server servicing a queue with Markovian arrivals and service times in first-in/first-out order. Other common queue types include M/M/Inf queues, which induce a fixed service delay on incoming requests but no queueing delay, M/M/c stations where the service time depends on the number of queued requests, and M/G/1 stations where service times follow a general (*i.e.* non-exponential) distribution [16].

In the general case, analyzing the performance of queueing networks involves solving for the stationary distribution of the underlying Markov chain. For non-trivial queueing networks, analyzing the full state-space of this system can quickly become prohibitive. Fortunately, a wide range of techniques exist for quickly solving or at least approximating the performance of many important subclasses of queueing models. The most well-known of these are product-form networks, networks of stations with restricted types (generally M/M/x stations and several variants) and service times (*i.e.* uniform service times across all request classes) that can be relatively quickly analyzed. In this paper, we restrict our models to such networks, though other more general formulation are also viable, as we discuss in Section 6. Interested readers are referred to the many excellent references on queueing systems for more detail [16, 15, 5, 12].

3. A QUEUEING MODEL OF MPI POINT-TO-POINT COMMUNICATION

Our model of MPI point-to-point communication performance is based on closed product-form queueing networks. As described in Section 2, these networks support limited types of queueing stations, queueing disciplines, and arrival and service time distributions. We use these stations to model key processing and communication components in the high performance systems, eliding system components that are not likely to influence communication performance to simplify the model. As necessary, each components can also be subdivided into additional components for further accuracy; we limit our models to the minimum number of stations necessary to capture the key performance tradeoffs we seek to study.

3.1 General Model Structure

Figure 1 shows the general structure of our models, which is similar to many previous point-to-point network queueing models [16]. This basic model is comprised of a closed network of two hosts with links with limited bandwidth and latency. Hosts are represented by multiple M/M/1 queues with one queue per core; when workloads (*e.g.,* MPI endpoints) are bound to specific cores, we use multi-class queueing models to capture this behavior. The network link itself is represented as two pairs of stations, one pair in each direction. The first queue in each pair is a delay (M/M/Inf) stations that induces wire time and switching costs, and second in each pair is an M/M/1 FIFO station that processes 32-byte flits at network bandwidth speed.

We make a number of key assumptions to restrict our model to product-form networks. First, we do not model the NIC transmit, receive, and DMA engines, and assume that they are part of the network link. We make this assumption because of challenges with message synchronization and pipelining—queueing models assume a request is processed at only one station at a time, so pipelining is typically modeled using multiple independent requests. The use of multiple independent requests, however, prevents modeling the synchronization that occurs when all elements are received.

In addition, we assume exponentially distributed service times for all stations and exponentially distributed message interarrival times. We also assume that an MPI receive is preposted for each incoming transmission. Finally, we assume that the time to match an incoming message is exponentially distributed. The first assumption is the largest simplification, particularly for BSP-style applications that queue bursts of message sends and SMP systems where multiple cores synchronize their activities; this assumption may be more reasonable, however, in emerging many-task programming models [21]. The second and third assumptions are generally true for scalable MPI programs, but there are important dynamics of how this relates to issues such as flow control that are not covered by this basic model. We discuss techniques for relaxing these assumptions in Section 6.

3.2 Modeling Key MPI Features

We construct a model for a particular scenario by determining (1) the number of times a communication request in a class *visits* each queueing station, and (2) the service

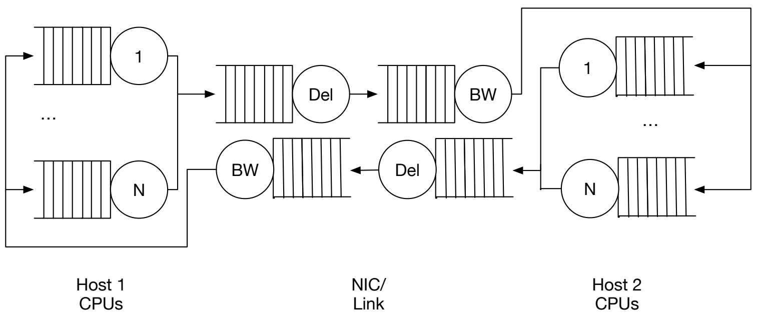

Figure 1: General Queueing Architecture of Model

Model Station	Service Time	Parameters
Host Core	1ns	Visit counts set based on message length for *matching delay*, eager message *copy bandwidth*, and per-message *protocol delays* costs for onload NICs.
Link delay	Offloaded protocol delays plus wire latency	Visit count set to the number of messages that traverse the link to process a request
Link bandwidth	Serial flit processing time	Visit count set based on the total number of flits that have to pass through a link to process request

Table 1: Queuing stations and parameters in model

time for a visit at each station. For example, in the case of network links, we set the M/M/1 queue service time for each direction of the network link to the time to process a 32-byte flit at full network bandwidth. Each message visits the queue based on the number of flits in the corresponding message. This allows small messages (*e.g.,* request-to-send (RTS) and clear-to-send (CTS) requests) to pass large messages (*e.g.,* data transfers) in the model, but a large message must finish all of its visits to a station prior to proceeding to the next station for processing. Table 1 lists the full set of stations and the parameters we use to set the service time and number of visits to each station.

Modeling Long and Short Protocols: As an example of the use of these parameters, consider a single rendezvous MPI transmission from host 1/core 4 to host 2/core 4. This request adds two visits to the outgoing delay station, one for the RTS and one for the data transmission. It also adds one visit to the return delay station for the CTS, a number of visits to each bandwidth station based on the size of the RTS, CTS, and data being sent, and visits on the receiving core to handle MPI matching costs. An eager message transmission, on the other hand, does not include the visits associated with the RTS or CTS, but adds additional visits to the receiving CPU to account for copying the message to its destination buffer. Additional visits on the receiving core can be used to model computation on the received data.

Modeling Protocol Onload vs. Offload: Similarly, protocol onload and offload tradeoffs can be modeling by adjusting service times and visit counts. In the case of protocol onload, we remove protocol processing delays from the NIC delay station and add extra per-message visits to core 1 on the transmitting and receiving hosts. We also reduce match delays on the receiving host because matching is typically overlapped with protocol processing on MPI onload NICs.

4. MPI PING-PONG AND MESSAGE RATE MODELING

To demonstrate of the feasibility of this formulation, we validate it's performance against the well-known MPI ping pong benchmark and use it to model MPI message rate, another important point-to-point communication characteristic. We model the performance of both standard MPI eager short-message protocols that require a copy on the receiving host and rendezvous protocols that avoid this copy using the techniques described in the previous section.

We use the model parameters shown in Table 2 in these cases. These parameters were determined by measuring the IB performance characteristics two Debian Linux nodes with integrated Mellanox ConnectX-3 cards running at QDR speeds, both running OpenMPI 1.4.5. We used `ibv_rc_pingpong` to measure peak one-way link bandwidth and half round-trip latency. As we run our tests on Mellanox offload network interfaces, it is not possible to directly measure the split between wire and protocol latency, so we split it approximately; in the offload model, these values are added together in series so the spliit does affect the correctness of the model. MPI match latency was measured using the difference between IB and OpenMPI 1-byte half round trip latency, and memory bandwidth is that specificed by the PCI Express bus in the nodes used in the test. We implement and solve the resulting model using the mean-value analysis solver in GNU Octave [11] *queueing* package [17].

Figures 2 and 3 show the modeled latency and bandwith bandwidth performance of the MPI ping-pong model with the parameters shown in Table 2. The extra cost of rendezvous transfers for large transfers compared to eager trans-

Parameter	Value
Bandwidth	3.787 GB/sec
Hardware Latency	1616 ns
Match Latency	271 ns
Memcpy bandwidth (single thread)	16 GB/sec

Table 2: Parameters used in model setup, measured on a x86 PCI Express 3.0 cluster node with a Mellanox InfiniBand QDR 4x NIC

fers are clear in these results. Similarly, rendezvous's bandwidth benefits over eager transfers are also clearly visible. Both of these results match generally expected performance results.

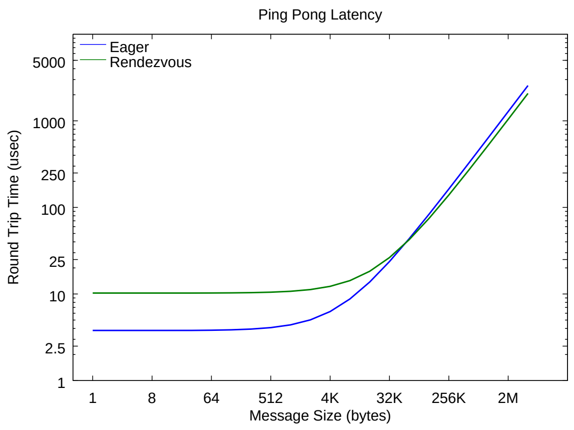

Figure 2: Modeled MPI Eager and Rendezvous Ping Pong Latency

To more carefully validate model performance, we then compared modeled ping pong latency and bandwidth to that measured using the OSU ping pong latency benchmark `osu_latency`. Matching the behavior of the OpenMPI version used, the model's crossover from eager to rendezvous protocols is set at 12KB. Figures 4 and 5 show the model results versus measured ping pong latency and bandwidth. Model results match measured values very closely, particularly for latency, generally validating model performance.

Similarly, we modeled MPI message rate performance since message rate is an increasing concern on upcoming many-core processors [3]. Unlike the previous case, transfers are unidirectional, and multiple sends are outstanding at a time, as in the Sandia message rate and bandwidth benchmarks [9]; visit counts in our model setup were changed correspondingly, and Figure 6 illustrates the natural decrease in message rate with increasing message size.

Building on these results, we also set out to model how changes in match list length impact MPI message rate performance. First, we modeled MPI match time using a simple linear model based partially on the results from previously-published many-core message rate studies [3]. Specifically, we modeled match delay in microseconds for a many-core

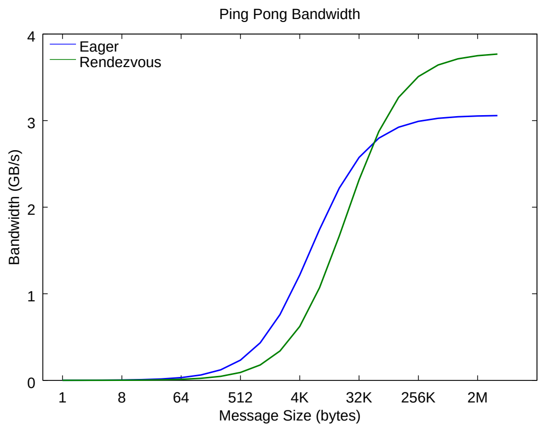

Figure 3: Modeled MPI Eager and Rendezvous Ping Pong Bandwidth

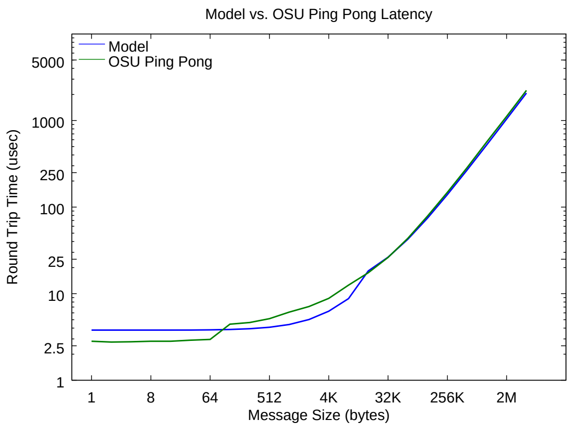

Figure 4: Modeled MPI Ping Pong Latency vs. Measured using `osu_latency` benchmark

system based on the results for ARM Cortex-A9 as

$$delay = 10 + 0.06 \times length \tag{1}$$

and match delay for heavyweight core systems based on the results for an Intel Ivy Bridge system as

$$delay = 1 + 0.004 \times length \tag{2}$$

These equations overestimate match delay for queue lengths less than 10, but are reasonable approximations as queue lengths increase.

Figure 7 shows the results of this model. The general shape of these results match those seen in past published work, and demonstrate both that our model can capture the approximate impact of fine-grain MPI features such as matching costs.

5. MANY-CORE MPI MESSAGE RATE AND PROGRESS TRADEOFFS

Following on this initial demonstration, next we used this model formulation to examine the potential impact of many-

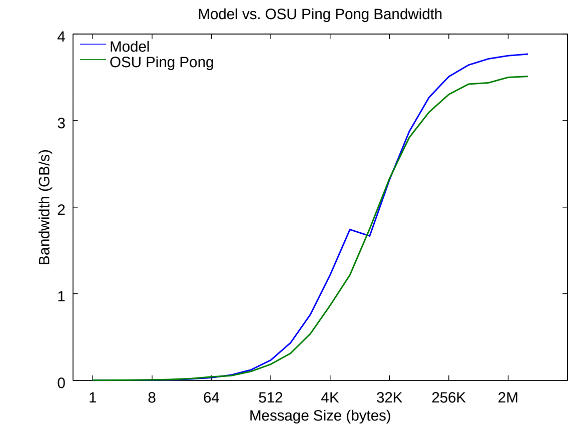

Figure 5: Modeled MPI Ping Pong Bandwidth vs. Measured using `osu_latency` benchmark

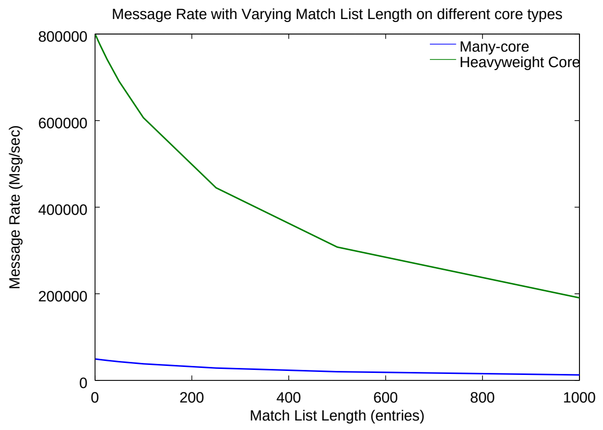

Figure 7: MPI Message Rate as a function of changing match rates. Matching delays are linearly approximated from match list length.

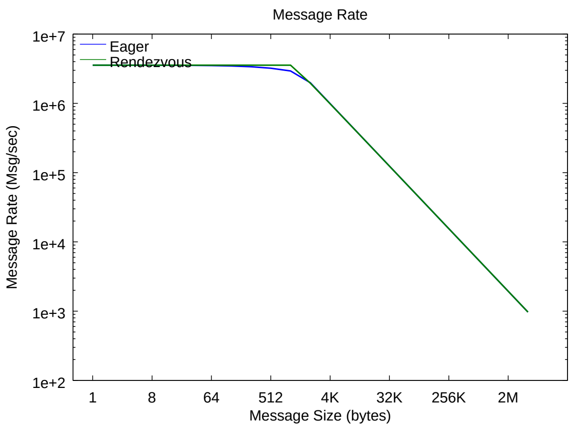

Figure 6: MPI Unidirectional Message Rate

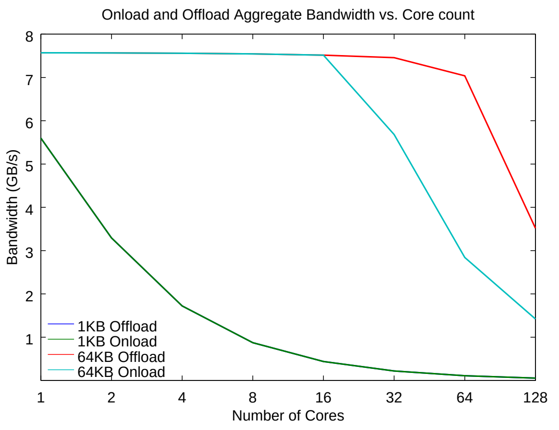

Figure 8: Bandwidth with increasing core count, decreasing message size, and all MPI progress funneled through core 1

core processors on MPI point-to-point performance. As described in Section 3, we use multi-class queueing models to bind traffic to specific core when appropriate. This allows, for example, protocol processing for onload NICs to be handled by one core, and the MPI-level progress core to be determined based on the sending and receiving core.

In this case study, we focused on one key performance tradeoff—the impact of different numbers of progress engines on MPI message rate performance. Most MPI many-core applications either avoid all message parallelism (`MPI_THREAD_SINGLE`) or funnel all MPI traffic through a single core (`MPI_THREAD_FUNNELED`) because of the performance problems associates with current multi-threaded MPI implementations. On many-core processors, this places the burden of MPI progress entirely on one core.

Figures 8 and 9 model the effect of using a single progress engine with a nominal matching delay of 100 nanoseconds on unidirectional bandwidth and message rate. In this case, as core count doubles, message size is halved but the number of messages in the system doubles. As a result, the same quantity of messaging traffic is present in the system in each case. Despite this, effective communication bandwidth drops quickly, particularly for onloaded NICs and with smaller initial message sizes. This occurs because a single core cannot keep up with the progress requirements of the additional cores despite the low per-message cost of MPI matching. The impact is more significant on onload cores, because the onloaded protocol processing costs borne by core 1 also reduce effective message rate.

We then modeled the impact of introducing additional progress engines into the system over a broad range of matching costs, as shown in Figure 10. In this case, message size is fixed at 4KB, 128 cores are used, and only offload NIC performance is modeled. Introducing additional progress engines, either as separate MPI processes or as additional endpoints [8] projects to significantly improve MPI message rate up to 16 cores handling MPI progress. These progress advantages must, however, be balanced with the number of cores needed by the application for computation.

6. DISCUSSION

As described in Section 3, we make a number of central assumptions to enable the use of product-form networks for MPI point-to-point communications. These assumptions

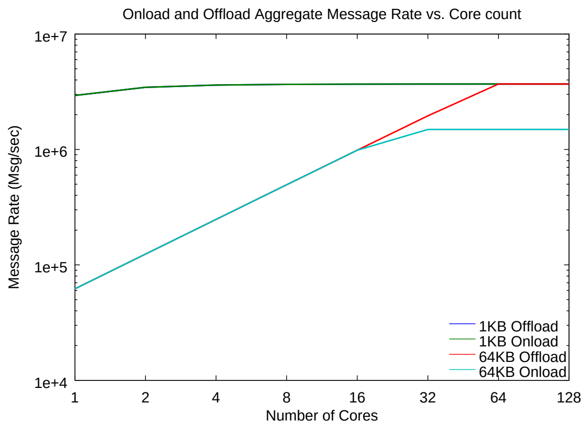

Figure 9: Message rate with increasing core count, decreasing message size, and all MPI progress funneled through core 1

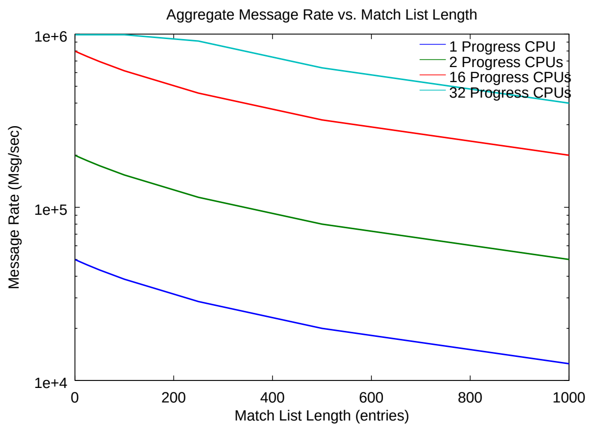

Figure 10: Message rate with increase match list length and differing number of MPI progress engines

limit the generality of our model. In this section, we discuss those limitations in more details, as well as potential approaches to removing or relaxing them.

6.1 Message pipelining

Message pipelining [19] is a key optimization for MPI point-to-point communications, allowing the sending host, receiving host, NIC, and switches to process portions of large messages in parallel. As discussed previously, however, the current model structure does not correctly model this. In particular, product-form models cannot model the synchronization that occurs after all message fragments in a pipeline arrive at a station. While bulk arrival queueing models do encompass this behavior, they are not easily analyzable.

Because of this, product-form models such as ours must *either* model pipelining effects or synchronization effects. We choose to focus on the later for the models in this paper because of the general structure of the benchmarks on which we focused. As a result, we simplified the underlying model to ammeliorate some of the inaccuracies this caused. However, for scenarios in which pipelining effects are important compared to synchronization costs, for example when large numbers of request are in the system, the alternative approach is likely preferable.

6.2 Barrier and Rendezvous Synchronization

Product-form models also struggle to model synchronization between multiple cores in other ways. For example, barrier synchronization between multiple stations is common in bulk-synchronous parallel applications and not directly analyzable in our approach. Similarly, important resource allocation actions such as pinning and registering pages with progress engines are also difficult to model with product form netwoks. However, a number of approximations exist to modeling synchronization. Models that include fork-join queues [22], for example, can be used to model and approximately solve for the performance of some simple barriers, for example multiple cores synchronizing prior to sending a single large message versus sending multiple smaller messages independently. Similarly, Petri networks and the varied techniques for solving them [5] can also be used to model the performance of more complex synchronization primitives, albiet at the cost of increased solution time.

6.3 Non-exponential service times

Finally, parallel programs frequently have service times and inter-arrival rates that are non-exponential. For example, carefully engineered systems may have service times that are deterministic instead of exponential, bursty, or heavy-tailed. Matrix analytic methods [12] provide a way of handling these models, though we have not yet attempted to apply them.

7. RELATED WORK

A large number of papers have sought to model parallel communication performance. However these efforts of focused primarily on *quantifying* point-to-point behavior and then using fitted parameters to reason about the performance of larger parallel constructs; the LogP model [7] mode and its variants (*e.g.,* [2]) are the most well-known of these. These models and similar efforts [13] use coarse messaging parameters, for example latency, communication/computation overlap, and message gap to derive the performance of applications and collective communication operations. Similarly, these models have also been used as the basis for simulations of MPI collective communication and application performance[1, 14].

While queueing models have not, to the best of our knowledge, been used for MPI performance analysis, they have been widely used for the analysis of networking and other computer systems, including the well-known work analyzing the ALOHA MAC protocol [18]. More recently, queueing models have been used for analyzing synchronization in multi-core systems [6], as well as a range of other constructs. These applications, recent advances in the area, and a desire to apply them to HPC systems were a key motivation behind the research described in this paper.

8. CONCLUSIONS AND FUTURE WORK

The importance of MPI communications for many-core architectures and examining offloaded versus onloaded networking has been explored in prior works [3, 10]. However, such work has thus far only examined existing hardware;

as such, it had limited utility for projecting future hardware capabilities or suggesting particular design points that would help enable efficient MPI communication for future exascale systems, particularly those based on many-core architectures.

We maintain that fine-grain communication system performance models will be essential to properly designing HPC communication systems for exascale computing system. Without such models, systems will not be able to effectively manage the complex tradeoffs resulting from, for example, large numbers of cores, memory bandwidth limits, dynamic power management, and network contention. New models that quantify such tradeoffs, however, could enable a new generation of adaptive, high-performance exascale communication systems.

This paper provides a first step toward the development of such fine-grain performance models. The goal of the model presented in this paper, as well as future work based on this model, is to enable the exploration of the complex communication design space and determine what approximate hardware resources are required for efficient communication. These models demonstrate the ability to capture key elements of MPI performance in analytical models, though do have important limitations that make their use in modeling MPI performance somewhat challenging.

A large number of directions for future work remain. Key next steps include quantifying MPI application and benchmark messaging workloads to quantify the extent to which our modeling assumptions are correct, and a full validation of these models against a broader set of MPI benchmarks. Beyond this, we plan to apply fork-join and other approximations to more fully study tradeoffs in synchronization, bandwidth, latency, and messaging progress in communication runtimes. This and other enhancements will help expand the model's ability to predict communication system needs for future hardware. Finally, we also envision a a broader approach that uses models of the form described in this paper along with similar models to systematically drive the development of next-generation communication libraries.

9. REFERENCES

[1] K. Al-Tawil and C. A. Moritz. Performance modeling and evaluation of MPI. *Journal of Parallel and Distributed Computing*, 61(2):202–223, 2001.

[2] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation. In *7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '95)*, pages 95–105, 1995.

[3] B. W. Barrett, R. Brightwell, R. Grant, S. D. Hammond, and K. S. Hemmert. An evaluation of MPI message rate on hybrid-core processors. *International Journal of High Performance Computing Applications*, 28(4):415–424, 2014.

[4] B. W. Barrett, S. D. Hammond, R. Brightwell, and K. S. Hemmert. The impact of hybrid-core processors on mpi message rate. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 67–71, New York, NY, USA, 2013. ACM.

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing networks and Markov chains - modeling and performance evaluation with computer science applications; 2nd Edition.* Wiley, 2006.

[6] S. Boyd-Wickizer, M. F. Kaashoek, R. Morris, and N. Zeldovich. Non-scalable locks are dangerous. In *Proceedings of the Linux Symposium*, pages 119–130, 2012.

[7] D. Culler, R. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '93)*, pages 1–12, San Diego, CA, 1993.

[8] J. Dinan, R. E. Grant, P. Balaji, D. Goodell, D. Miller, M. Snir, and R. Thakur. Enabling communication concurrency through flexible MPI endpoints. *International Journal of High Performance Computing Applications*, 28(4):390–405, 2014.

[9] D. Doefler and B. W. Barrett. Sandia MPI microbenchmark suite (SMB). Technical report, Sandia National Laboratories, 2009.

[10] M. G. Dosanjh, R. Grant, P. Bridges, and R. Brightwell. Re-evaluating network onload vs. offload for the many-core era. In *Proceedings of the 2015 IEEE International Conference on Cluster Computing (Cluster 2015).* IEEE, September 2015.

[11] J. W. Eaton, D. Bateman, and S. Hauberg. *GNU Octave Manual Version 3.* Network Theory Ltd., 2008.

[12] M. Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action.* Cambridge University Press, New York, NY, USA, 1st edition, 2013.

[13] T. Hoefler, W. Gropp, R. Thakur, and J. L. Traeff. Toward Performance Models of MPI Implementations for Understanding Application Scaling Issues. In *Recent Advances in the Message Passing Interface (EuroMPI'10)*, volume LNCS 6305, pages 21–30. Springer, Sep. 2010.

[14] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the Influence of System Noise on Large-Scale Applications by Simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010.

[15] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* Wiley New York, 1991.

[16] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1984.

[17] M. Marzolla. The `qnetworks` toolbox: A software package for queueing networks analysis. In K. Al-Begain, D. Fiems, and W. J. Knottenbelt, editors, *Analytical and Stochastic Modeling Techniques and Applications, 17th International Conference, ASMTA 2010, Cardiff, UK, Proceedings*, volume 6148

of *Lecture Notes in Computer Science*, pages 102–116. Springer, June14–16 2010.

[18] N. Meisner, J. Segal, and M. Tanigawa. An adaptive retransmission technique for use in a slotted-ALOHA channel. *IEEE Transactions on Communication*, COM-28:1176–1178, 1980.

[19] A. Rodrigues, K. Wheeler, P. Kogge, and K. Underwood. Fine-grained message pipelining for improved mpi performance. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–9. IEEE, 2006.

[20] S. Sridharan, J. Dinan, and D. D. Kalamkar. Enabling efficient multithreaded MPI communication through a library-based implementation of mpi endpoints. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 487–498. IEEE, 2014.

[21] D. T. Stark, R. F. Barrett, R. E. Grant, S. L. Olivier, K. T. Pedretti, and C. T. Vaughan. Early experiences co-scheduling work and communication tasks for hybrid mpi+ x applications. In *Proceedings of the 2014 Workshop on Exascale MPI*, pages 9–19. IEEE Press, 2014.

[22] E. Varki. Mean value technique for closed fork-join networks. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '99, pages 103–112, New York, NY, USA, 1999. ACM.