Toward Message Passing for a Million Processes: Characterizing MPI on a Massive Scale Blue Gene/P*

P. Balaji[1] A. Chan[1] R. Thakur[1] W. Gropp[2] E. Lusk[1]

[1]Math. and Comp. Science,
Argonne National Laboratory,
{balaji, chan, thakur, lusk}@mcs.anl.gov

[2]Dept. of Computer Science,
U. Illinois, Urbana Champaign,
wgropp@illinois.edu

Abstract

High-end computing (HEC) systems have passed the petaflop barrier and continue to move toward the next frontier of exascale computing. Systems with hundreds of thousands of cores are already available and upcoming exascale capable systems are expected to comprise more than a million processing elements. As companies and research institutes continue to work toward architecting these enormous systems, it is becoming increasingly clear that these systems will utilize a significant amount of shared hardware between processing units, including shared caches, memory management engines and network infrastructure. Thus, understanding how effective current message passing and communication infrastructure is in tying these processing elements together, is critical to making educated guesses on what we can expect from such future machines. Thus, in this paper, we characterize the communication performance of the message passing interface (MPI) implementation on 32 racks (131,072 cores) of the largest Blue Gene/P (BG/P) system in the world (80% of the total system size). Our studies show various interesting insights into the communication characteristics of MPI on the BG/P.

1 *Introduction*

As we move into an era of petaflop computing, and look forward to multi-petaflop and exaflop computing, modern high-end computing (HEC) systems are rapidly increasing in size. With processor speeds no longer doubling every 18-24 months due to the exponential increase in power consumption and heat dissipation, modern HEC systems tend to rely lesser on the performance of single processing units, but rather try to extract parallelism out of a massive number of processing elements. IBM Blue Gene/L [5] was one of the early supercomputers to follow this architectural model and was soon followed by other systems such as the Blue Gene/P (BG/P) [9] and SiCortex [4].

Today, large systems using these architectures already scale to hundreds of thousands of processing elements. With plans underway for exascale systems to emerge within the next decade, it is expected that we will soon have systems that comprise more than a million processing elements. As researchers work toward architecting these enormous systems, it is becoming increasingly clear that these systems will utilize a significant amount of shared hardware. This includes shared caches, shared memory and memory management devices, and shared network infrastructure. One of the primary challenges in such architectures, that use a massive quantity of modestly powerful processing units instead of a few very powerful processing units, is their capability to tie these units together into a tightly coupled network fabric that allows them to appear as one fast supercomputer. This challenge is even more formidable given the increasing amount of shared hardware in such systems. Thus, understanding how effective the current message passing and communication infrastructure is in tying these processing elements together is critical to making educated guesses on what we should expect from future exascale machines that follow a similar architecture.

In this paper, we characterize the communication performance of the Message Passing Interface (MPI) on 32 racks (131,072 cores) of the largest Blue Gene/P system in the world (80% of the total system size). Our studies include tests that stress the shared hardware in the system. The paper documents several interesting observations including the impact of swap-free memory, DMA engine utilization model, impact of multiple network hops and network congestion behavior. We also demonstrate the aggregate effect of all these observations using the communication kernel of the NRL Layered Ocean model (NLOM) [14]—a simulation model the allows scientists to understand the behavior of semi-enclosed seas, major ocean basins and the global ocean.

2 *BG/P Hardware and Software Stacks*

Here we describe the hardware and software stacks of BG/P.

2.1 *BG/P Hardware Architecture*

As shown in Figure 1, the BG/P uses a 4-core architecture with each core having a separate L2 cache and a semi-distributed L3 cache (shared between two cores). Each node is connected to five different networks [10]. Two of them, 10-Gigabit Ethernet and 1-Gigabit Ethernet with JTAG interface, are used for file I/O and system management. The other three are used for MPI communication as described below.

3-D Torus Network: This network is used for MPI point-to-point and multicast operations and connects all compute nodes to form a 3-D torus. Thus, each node has six nearest-neighbors. Each link provides a bandwidth of 425 MB/s per direction, for a total bi-directional bandwidth of 5.1 GB/s. As shown in Figure 1, though each node has six bidirectional links on each node, there is only one shared DMA engine.

Global Collective Network: This is a one-to-all network for

*This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy under contract DE-AC02-06CH11357 and in part by the Department of Energy award DE-FG02-08ER25835

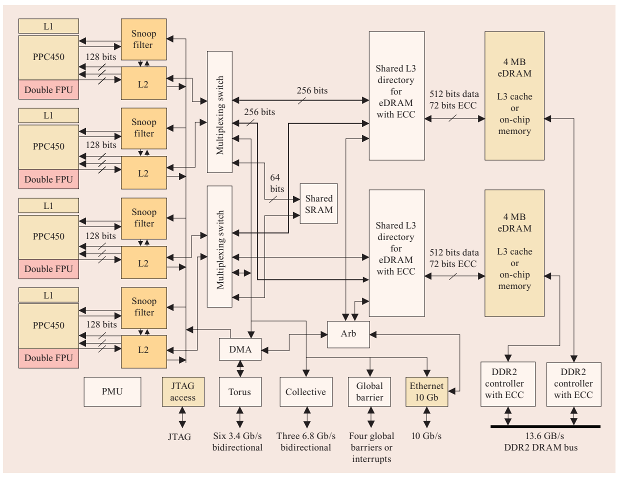


Figure 1: BG/P Architecture [9]

compute and I/O nodes used for MPI collective communication (for regular collectives with small amounts of data) and I/O services. Each node has three links to this network (total of 5.1 GB/s bidirectional bandwidth).

Global Interrupt Network: This is an extremely low-latency network that is specifically used for global barriers and interrupts. Messaging on this network is extremely scalable. For example, the global barrier latency of a 72K-node partition is approximately 1.3μs.

The compute cores in the nodes do not handle packets on the torus network; a DMA engine on each node offloads most of the network packet injecting and receiving work, which enables better overlap of computation and communication. However, the cores directly handle sending/receiving packets from the collective network.

2.2 *BG/P Software Architecture*

BG/P is designed for multiple programming models. The Deep Computing Messaging Framework (DCMF) and the Component Collective Messaging Interface (CCMI) are used as general purpose libraries to support different programming models [13]. DCMF implements point-to-point and multisend protocols. The multisend protocols connect the abstract implementation of collective operations in CCMI to targeted communication networks. DCMF provides three types of message-passing operations: two-sided send, multisend and one-sided get, all three with nonblocking semantics.

The MPI implementation on BG/P is MPICH2 [19] layered on top of DCMF. MPICH2 provides an internal abstraction layer, called the abstract device interface (ADI), that allows it to be implemented and tuned for new platforms with modest effort, while still retaining most of its upper-level code, including the ROMIO implementation of MPI-IO and the MPE profiler. IBM wrote, and contributed back to the open-source MPICH2 code base, a new implementation of the ADI, called `dcmfd`. This enables MPICH2 to run efficiently on BG/P and re-implements many of the collective communication functions to take advantage of the special networks and hardware features of BG/P.

3 *Experimental Analysis*

In this section, we perform several experiments to understand the communication characteristics of MPI on BG/P.

3.1 *Two-process Point-to-point Benchmarks*

This section characterizes two-process benchmarks on BG/P.

3.1.1 *Inter-node Performance*

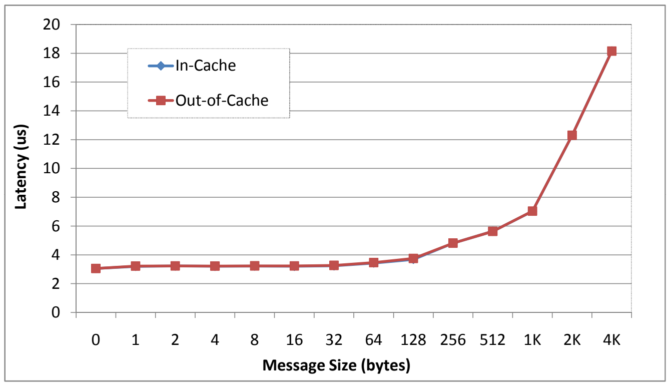
Figure 2(a) illustrates the one-way ping-pong latency achieved by MPI on BG/P between two nodes separated by a single network hop. In this experiment, the sender sends a message of size S to the receiver. On receiving this message, the receiver sends back another message of the same size to the sender. This is repeated several times and the total time averaged over the number of iterations, which gives the average round-trip time. The ping-pong latency reported here is one half of the round trip time, i.e., the time taken for a message to be transferred from one node to another.

The figure shows two legends: in-cache and out-of-cache. For "in-cache", the same buffer is used for each communication iteration, so the buffer is always in cache. Conversely, for "out-of-cache", a different buffer is used for each iteration, causing the buffer to be out-of-cache each time. We notice that there is no difference between in-cache and out-of-cache performance, both achieving about 2.8 μs small message latency. This is because of the memory management functionality of BG/P which does not maintain any virtual address swap space. Thus, all its virtual address space is always pinned to physical memory pages. Therefore, unlike other cluster network interconnects such as InfiniBand [6] and Quadrics [21], BG/P does not have to perform any separate memory pinning before communication and the DMA engine can directly communicate from any buffer in a zero-copy manner. Consequently, the processor does not have to touch the data for any processing, thus causing no degradation in performance irrespective of whether the data being communicated is in cache or not.

Figure 2(b) illustrates the unidirectional bandwidth. In this experiment, the sender sends a message of size S a number of times to the receiver. On receiving all the messages, the receiver sends back one small message to the sender, indicating that it has received the messages. The sender calculates the total time, subtracts the one-way latency of the message sent by the receiver, and based on the remaining time, calculates the amount of data it had transmitted per unit time. Like the latency experiment, we notice that there is no impact on bandwidth either based on whether the data is in cache or not; both forms achieve about 3 Gb/s unidirectional bandwidth for large messages.

3.1.2 *Intra-node Performance*

Figures 3(a) and 3(b) show MPI ping-pong latency and unidirectional bandwidth between cores on the same node. Communication performance is measured between core 0 and one other core as indicated by the legend. We make several observations



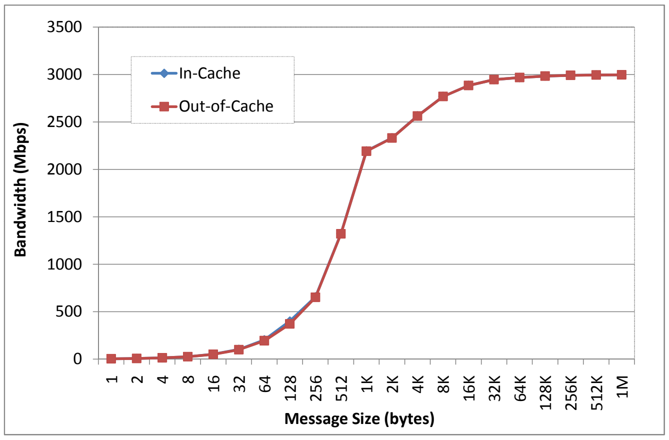
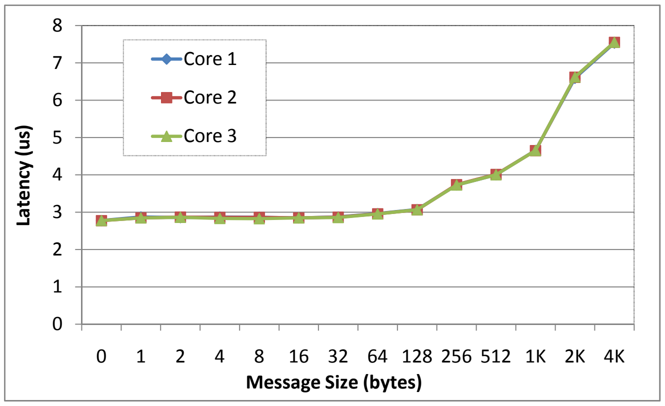


Figure 2: Inter-node Performance: (a) One-way Latency; (b) Unidirectional bandwidth



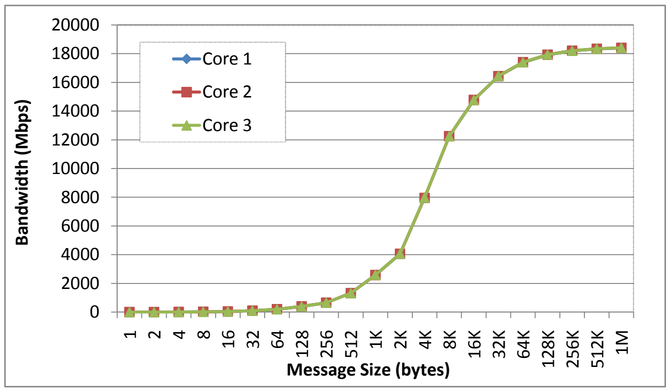


Figure 3: Intra-node Performance: (a) One-way Latency; (b) Unidirectional Bandwidth

in these two experiments. First, for ping-pong latency, we notice no performance difference irrespective of which two cores communicate. Second, the intra-node and inter-node latencies (Figures 3(a) and 2(a)) are identical (about 2.8 μs) for small messages. These two observations have the same underlying reason: the processing power of each core on the BG/P is only a modest 850 MHz; so unlike fast Intel and AMD processors, the time taken for memory copies is much higher on such processors. Accordingly, instead of using the processor for shared-memory communication, BG/P uses the hardware DMA engine for both intra-node and inter-node communication. Thus, there is no difference in performance in the two. Due to the same reason, it does not matter, with respect to performance, which two cores perform communication.

For the unidirectional bandwidth, we again notice no performance difference based on which two cores communicate due to the same reason as above. We also notice that the intra-node communication bandwidth (Figure 3(b)) is about six-fold higher than the inter-node bandwidth (Figure 2(b)). This difference is due to the capability of the DMA engine. As mentioned in Section 2.1, the DMA engine is shared between all the six torus links of the node. Thus, in order to be able to drive all six bidirectional links, it has to be capable of six times the single-link communication bandwidth for data going out as well as for data coming in. In an intra-node communication test, the data from one process' address space has to go down to the DMA engine and come back up to the second process' address space, which the engine can perform six times faster than what an inter-node link can support.

3.1.3 *Impact of Hops on an Idle Network*

Figure 4 shows the impact of the system size on communication latency by performing the inter-node latency test using the two farthest nodes in the system. Thus, as the system size increases, the number of hops the message has to traverse also increases. As shown in the figure, the system size has a large impact on communication latency, especially for small and medium-sized messages. For example, for a zero-byte message, the performance degradation is close to 100% when the system size changes from 4 to 131,072 cores. Even for medium-sized messages of up to 1 KB, the impact is still 40%, which is significant. This illustrates that for latency-sensitive applications, the placement of the processes can play a significant role in performance as how far apart they are can determine their communication performance. For large messages, however, the impact of number of hops is minimal.

We performed a similar experiment with the bandwidth test, but did not notice any impact on performance (less than 3%). This is expected as messages are pipelined across network hops; thus

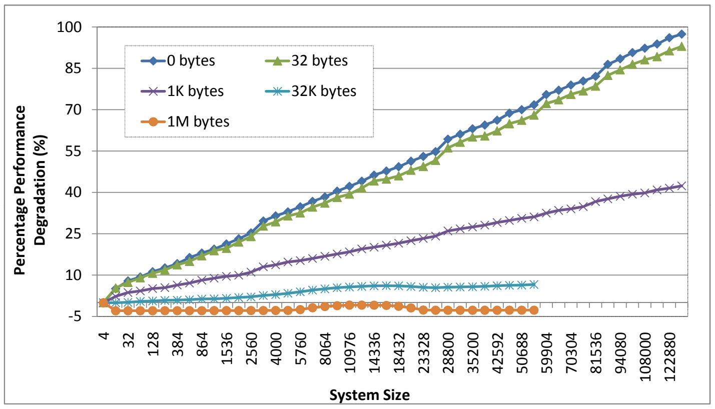


Figure 4: Impact of Number of Hops on One-way Latency

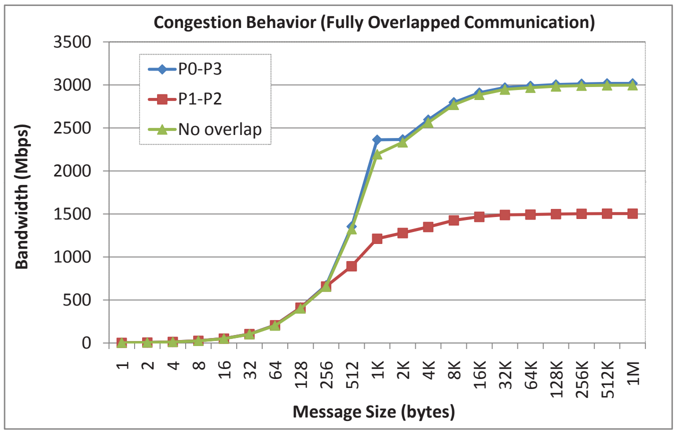


Figure 5: Network Congestion Behavior

the number hops should not matter for streaming communication, such as the bandwidth test.

We also analyzed the impact of network hardware sharing at each hop. This experiment was designed to analyze to what extent flow-through data (i.e., data that passes through a particular node on the torus, but is neither sourced at or destined to this node) utilizes the network hardware on each hop. Specifically, while a heavy amount of traffic is flowing through a node, we performed an intra-node bandwidth test on the same node. Since intra-node communication utilizes the DMA engine (as described in Section 3.1.2), if there is sharing of the DMA engine with the flow-through data, bandwidth performance should suffer. Our experiments revealed no such impact showing that flow-through data has other dedicated hardware and does not use the node's DMA engine. A similar test was done for inter-node bandwidth as well, but utilizing a different torus link for the bandwidth test than the one used for the flow-through data; no performance impact was noticed for that either.

3.2 *Multi-process Point-to-point Benchmarks*

This section characterizes multiprocess point-to-point communication for MPI on BG/P.

3.2.1 *Network Congestion Behavior*

In this section, we study the communication behavior on BG/P in the presence of network congestion by pushing multiple streams of data on the same link and measuring the performance achieved by each data stream. We pick four processes on a full torus system partition that are contiguously located along a single dimension (say P0, P1, P2 and P3). These four processes form two pairs, with each pair performing the bandwidth test.

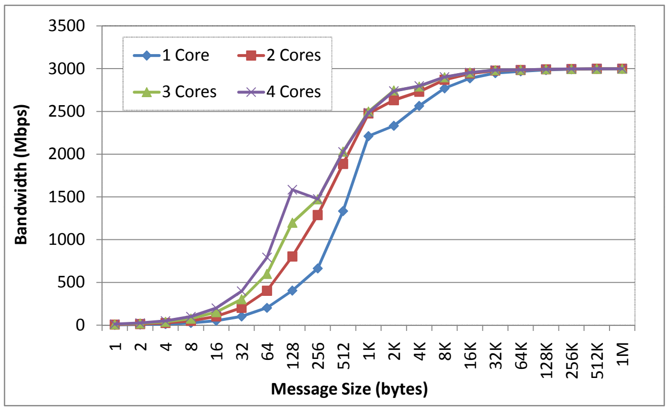
In the first experiment (Figure 5), P0 sends data to P3 (which takes the route P0–P1–P2–P3) and P1 sends data to P2 (which takes a direct one hop route, P1–P2). Thus, the link connecting P1 and P2 is shared for both communication streams. As shown in the figure, we see that the communication between P0 and P3 (legend "P0-P3") achieves the same bandwidth as an uncongested link (legend "No overlap") illustrating that the link congestion has no performance impact on this stream. However, for the communication between P1 and P2 (legend "P1-P2"), there is a significant performance impact. The reason for this asymmetric performance for these two streams is related to the congestion management mechanism of BG/P. Like most other networks, BG/P uses a sender driven data-rate throttling mechanism to manage network congestion. Specifically, when the sender is trying to send data, if the immediate link on which data needs to be transmitted is busy, the sender throttles the sending rate. On the other hand, for flow-through data the sender is not directly connected to the congested link and hence cannot "see" that the link is busy. Thus, there is no throttling for flow-through data causing it to achieve high-performance, but at the expense of other flows.

Another observation we make is that there are several paths between the pairs P0-P3 and P1-P2 that do not overlap with each other. However, the loss in performance for the P1-P2 pair illustrates that none of these additional paths are utilized and data is always sent in a statically pre-configured path.

The second experiment we performed is similar to the previous one, but using P0-P2 (routed as P0–P1–P2) and P1-P3 (routed as P1–P2–P3) as the process pairs. Thus, both flows have P1–P2 as the common link, and both flows are partially congested by each other. The performance observations are similar to the previous experiment, with the P0–P1–P2 achieving peak bandwidth, and P1–P2–P3 achieving a throttled bandwidth.

3.2.2 *Multistream Bandwidth*

The multistream bandwidth test is similar to the unidirectional bandwidth test described in Section 3.1.1, except that instead of just two processes performing the test, multiple pairs of processes perform the same test. Specifically, since each node is equipped with four cores, the test allows multiple cores on the node to participate in the communication. Thus, for the case of N cores, there are N streams of communication between the same two nodes. The aggregate bandwidth of all flows is computed and reported in Figure 6(a). As shown in the figure, the peak bandwidth achieved in all cases is the same. This is expected, as irrespective of the number of flows, the performance will eventually be limited by the link bandwidth. However,



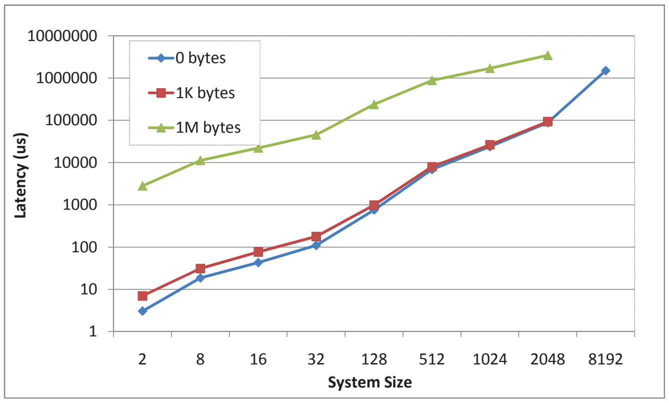


Figure 6: (a) Multi-stream Communication; (b) Hot-spot Communication

we notice that for medium-sized messages, the performance difference between just one core performing communication vs. multiple cores being involved in communication is close to twofold in some cases. Since the communication message sizes for many applications are in this range, this experiment gives a strong indication to application developers that utilizing multiple cores for communication can be significantly helpful especially while using low-frequency processing cores.

3.2.3 *Hot-spot Communication*

In this section, we measure the performance of hot-spot communication, where a single "master" process performs a latency test with a group of "worker" processes, thus forming a communication hot-spot. This test is designed to emulate master-worker kind of communication models. Figure 6(b) illustrates the average latency noticed by each worker processes for different message sizes over a range of system sizes (log-log plot). For all message sizes, we see an exponential increase in the hot-spot latency with increasing system size. This is attributed to the congestion that occurs when multiple messages arrive via the limited number of links surrounding a single master process. As the system size increases, more and more messages are pushed to the same process, further increasing congestion and causing significant performance loss.

In summary, the flat network topology of BG/P is not well suited for master-worker kind of communication, especially when the messages being communicated are large. Our measurements reveal that the system size at which performance begins to degrade is very small. For applications using such a communication pattern, hierarchical master-worker communication can alleviate bottlenecks in some cases, but can have serious performance constraints when scaled to very large sizes.

3.2.4 *Fan Communication*

In this section, we measure the performance of fan-based communication where a node communicates with its six physical neighbors that are directly connected along the links of the 3D torus. The fan-in test measures the process' capability to receive data from its neighbors and the fan-out test measures the process' capability to send data to its neighbors. Figures 7(a) and 7(b) show the fan-in and fan-out performance measurements, respectively, with increasing number of neighbors communicated with. As the number of neighbors increase, the overall performance increases in general. For the fan-out test, the peak performance achieved is about 18,000 Mb/s, which is close to the maximum performance capability of the DMA engine, as illustrated in Section 3.1.2. However, for the fan-in test, the peak performance saturates at only 13,000 Mb/s. This shows that the data-receiving path of the stack has more overhead compared with the sending path. Thus, one process sending data to multiple processes is expected to achieve better performance as compared to one process receiving data from multiple processes.

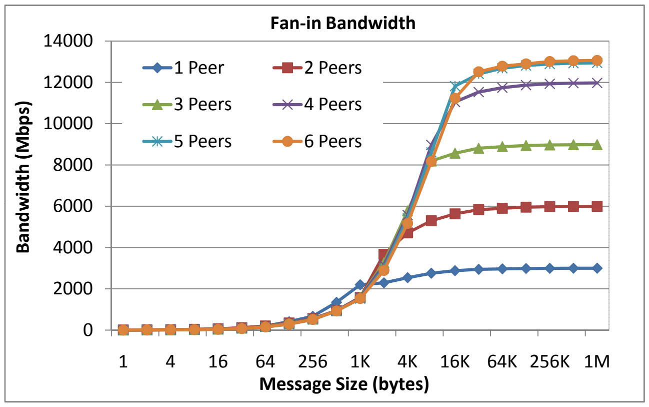
3.3 *Collective Communication*

In this section, we evaluate MPI collective communication.

3.3.1 *MPI_Barrier*

Figure 8(a) shows the performance of `MPI_Barrier` for different communicators with increasing system size: `MPI_COMM_WORLD`, dup of `MPI_COMM_WORLD` and a split communicator containing all processes in `MPI_COMM_WORLD` except the last process. As shown in the figure, both `MPI_COMM_WORLD` and a direct dup of `MPI_COMM_WORLD` perform identically. However, for a non-standard communicator such as `MPI_COMM_WORLD` without the last process, the performance is significantly worse. This is because communication for standard communicators such as `MPI_COMM_WORLD` is handled in hardware using the global interrupt network described in Section 2.1. For non-standard communicators, however, the barrier takes place in software, which has significantly higher overhead. We also notice a large variation in the barrier time based on the system size. Some of this is attributed to the system topology as different system sizes use different torus topologies. The rest is attributed to the software stack itself.

Figure 8(b) shows the performance of multiple parallel barriers happening on the same set of nodes. Specifically, the processes on core 0 of all nodes perform a barrier while the processes on core 1 of all nodes perform another parallel barrier, and so on. Since all the barriers share the same physical network, they might interfere with each other causing performance loss. We



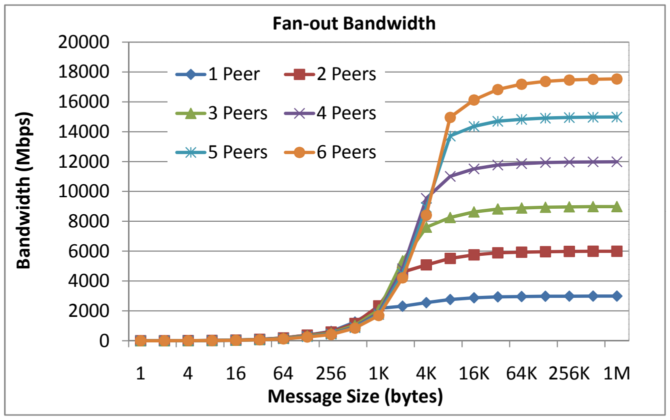
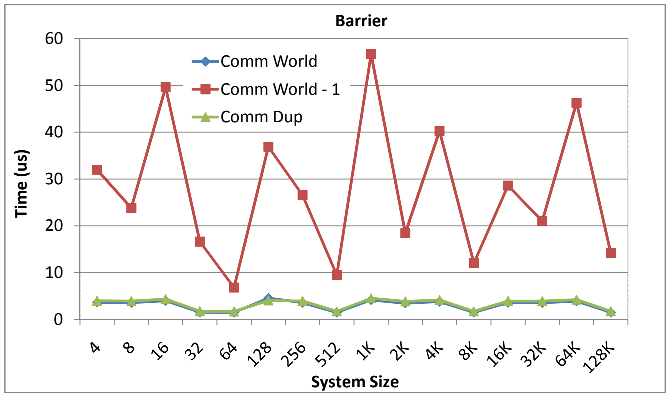


Figure 7: Fan Tests: (a) Fan-in; (b) Fan-out



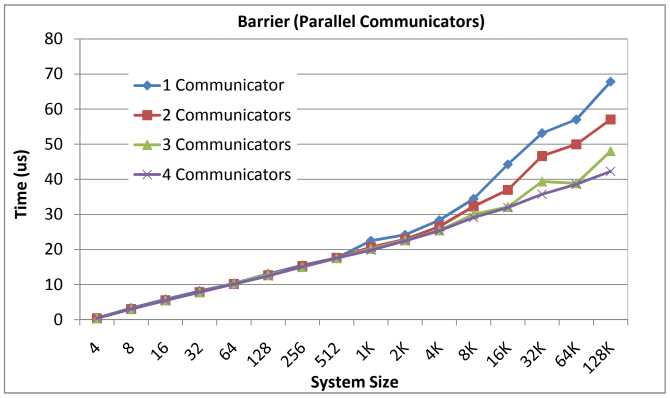


Figure 8: Barrier Performance: (a) Variance with Communicators; (b) Parallel Barriers

notice that for small system sizes, this interference is minimal. However, as the system size increases, we notice a counter-intuitive behavior—the average barrier time decreases with increasing number of parallel communicators! This behavior is attributed to the potential for software optimizations with parallel barriers. That is, with multiple parallel barriers occurring on the same set of nodes, the network stack has an opportunity to perform message coalescing. This allows the *average time* of the barrier to reduce as the information equivalent to multiple messages is carried out in a single message. In fact, for a system size of 131,072 cores, we notice that the interference actually causes a performance improvement of nearly 75%.

3.3.2 *MPI_Bcast*

We evaluated `MPI_Bcast` for the three different communicators described in Section 3.3.1 and found that `MPI_Bcast` on standard communicators performs nearly 10-fold better than non-standard communicators on a system size of 131,072 cores (Figure 9). Such performance difference can be critical for many application developers, since many scalable applications do not perform operations on `MPI_COMM_WORLD` in the performance critical path; instead they break up processes into smaller communicators (such as a Cartesian map) and perform operations on these smaller communicators.

Figures 10(a) and 10(b) show the performance of multiple parallel broadcasts for message sizes of 4 bytes and 16 KB, respectively. This experiment is similar to the one described in Section 3.3.1, except that it uses `MPI_Bcast` instead of an `MPI_Barrier`. For a 4-byte broadcast, we see that the trend is similar to `MPI_Barrier`. That is, as the number of parallel communicators increases, the average time taken by the broadcast reduces due to message coalescing. However, for a 16 KB broadcast, we see a trend reversal—performance degrades as the number of parallel communicators increases. This is because, for large messages, there is no real possibility for message coalescing. However, since the physical link is shared, this can result in communication interference leading to per-

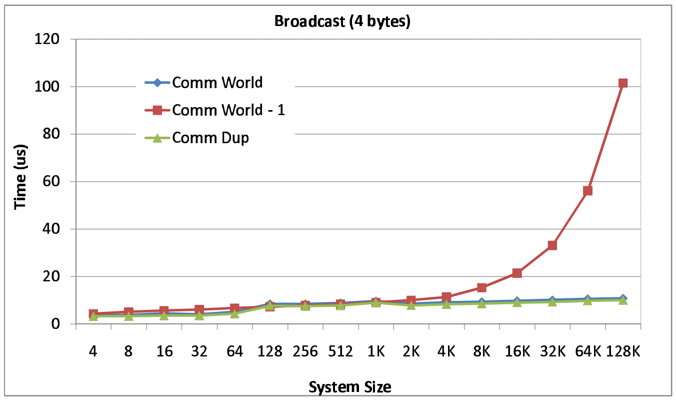
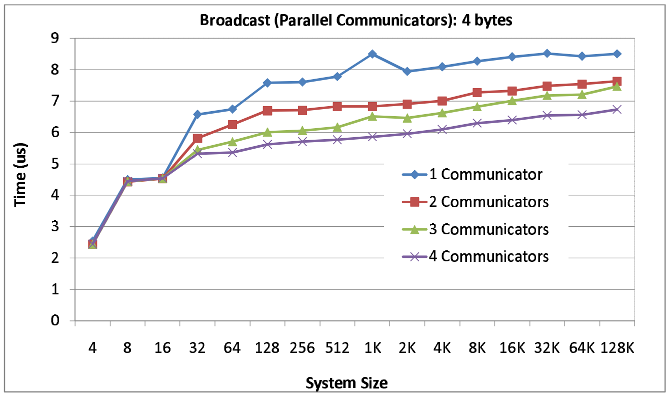


Figure 9: Broadcast on Different Communicators



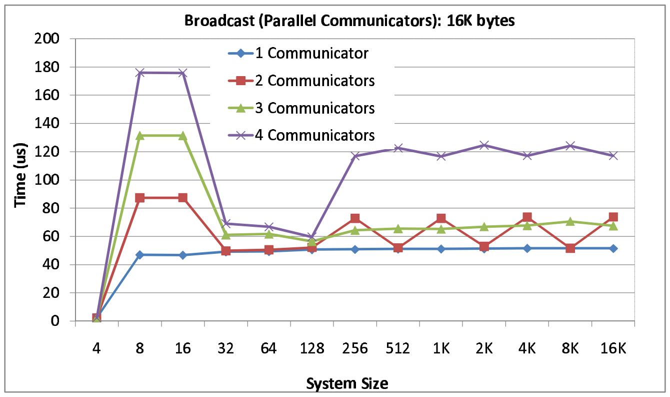


Figure 10: Parallel Broadcast Performance: (a) 4 byte message; (b) 16 KB message

formance loss. As the system size increases to 16K processes, we see a performance degradation of threefold going from one communicator to four.

3.3.3 *MPI_Allreduce and MPI_Allgather*

Figure 11(a) shows the performance of multiple parallel `MPI_Allreduce` operations. Unlike, barrier and broadcast, we notice that the performance of allreduce does not vary with multiple parallel communicators even for a 16 KB operation. This is because at each intermediate node, `MPI_Allreduce` has to process the incoming data, which is a bigger bottleneck than data communication itself. Thus, the communication interference is not visible in this operation.

Figure 11(b) shows the performance of multiple parallel `MPI_Allgather` operations. In this test, we see that even for a 4-byte Allgather, there is significant communication interference as the system size increases. This is because `MPI_Allgather` is an accumulative operation where the total data size increases with system size. Thus, even for a 4-byte `MPI_Allgather`, a 131,072-core system can cause very large messages, and consequently communication interference.

In summary, our experiments with parallel execution of collective operations show that the performance of an operation as perceived by real applications can be significantly different from what usual micro-benchmarks indicate, because of its interference with other communication in the system. Many applications divide processes into small groups, and each group communicates within itself. However, because of such communication interference, these applications might suffer from unexpected communication penalties.

3.4 *Process Mapping Effects on NLOM*

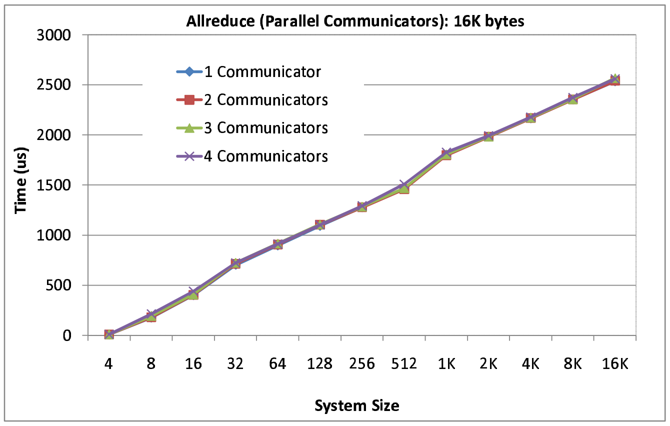
The NRL Layered Ocean Model (NLOM) [14] simulates semi-enclosed seas, major ocean basins, and the global ocean. The current implementation of the model uses tiled data-parallel programming style. Its general nature allows implementations in various programming models including MPI, OpenMP, Co-Array Fortran, and shared memory. This makes NLOM a good candidate for benchmarking both hardware and the associated communication software. The HALO benchmark simulates an NLOM 2-D exchange for an NxN sub-domain for different values of N. HALO puts a premium on low latency, much as NLOM as a whole does. In general, Halo exchanges are important operations whenever domain decomposition is used, but HALO can also be treated as a generic low-level communication benchmark. Small N performance is dominated by latency, and large N by bandwidth.

In this section, we analyze the effect of process mapping on the performance achieved by HALO. Several parameters affect the performance of HALO; these include (a) application specific parameters (such as whether the messages communicated are in cache or not, and how much intra-node vs. inter-node communication it performs) and (b) where exactly each application process is on the system and which other processes it communicates with (this determines the number of hops the messages have to traverse, how much congestion they create in the network and how much interference they have with other parallel communication in the system). Thus, varying the process mappings allows us to observe the extent of the overall effect of these parameters from an end-user's perspective.

Figure 12 illustrates the overall performance of HALO for different process mappings (XYZT, TXYZ, ZYXT, and TZYX) and system sizes (16K and 128K processes). Different mappings indicate how MPI ranks are allocated, e.g., XYZT indicates that ranks are ordered first with respect to the X-axis on the 3D torus, then Y-axis, and so on. T-axis refers to the cores within the node. As shown in the figure, these mappings can have up to twofold impact for a system size of 16K processes. As the system size increases to 128K processes, this impact increases to up to threefold. This indicates that, as the system sizes keep growing, such mapping will become even more important. In general, which mapping is the best is not a trivial question to answer as it depends on a number of parameters including several of those we described in this paper, as well as many others including the characteristics of the application.

4 *Related Work and Discussion*

There has been a significant amount of prior work related to understanding the performance characteristics of MPI on different architectures [17, 16, 15, 12, 8, 7]. However, this prior work



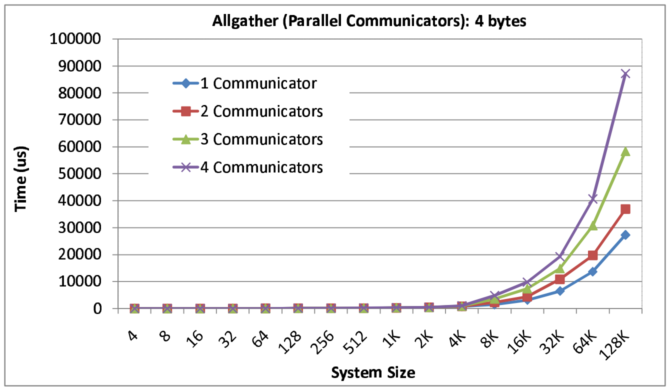
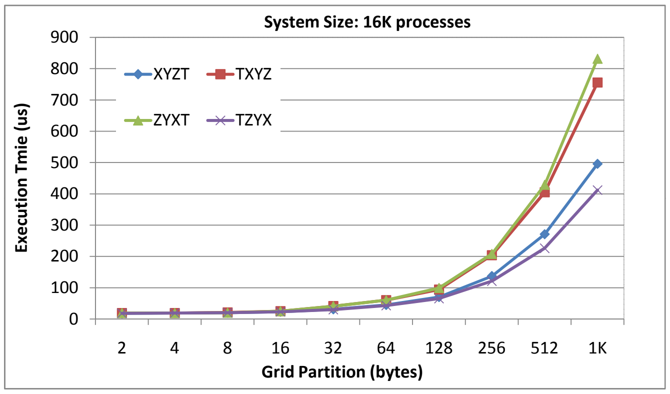


Figure 11: Parallel Collective Performance: (a) Allreduce (16 KB message); (b) Allgather (4 byte message)



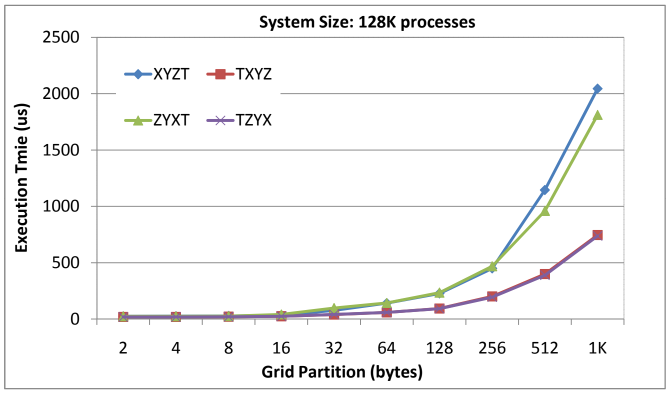


Figure 12: Nearest Neighbor Performance: (a) 16K processes; (b) 128K processes

primarily lags with respect to characterizing MPI on the scale that we study in this paper. Specifically, as we reach out toward exascale-capable systems in the next decade, there is no clear understanding so far on what can be expected from the massive parallelism that is available and the potentially huge amount of hardware sharing that is quickly becoming common with multi-core architectures, SMTs and flat networks. Our work attempts to bridge this gap.

Recently, there has also been interest in trying on understand what the most prominent parallel programming model for exascale systems would be. There has been work in extending MPI itself [18] as well as other models including UPC [1], Co-Array Fortran [2], Global Arrays [3], OpenMP [20] and hybrid programming models (MPI + OpenMP [11], MPI + UPC). Thus, to decouple ourselves from this aspect, while this paper uses MPI as a vehicle for evaluation, the studies bring out interesting observations in the low-level network communication of BG/P that are relevant to most prominent programming models and not just MPI.

In summary, this paper extends on existing prior work and brings out interesting performance aspects that are already true for current large-scale systems and will only become more prominent and visible for larger systems. Thus, we believe this work would be an interesting and highly relevant contribution to high-end computing research.

5 *Conclusions and Future Work*

In this paper, we characterized the communication performance of MPI on 32 racks (131,072 cores) of the largest Blue Gene/P system in the world (80% of the total system size). Our studies included benchmarks that stressed the shared hardware on the system. We identified various interesting insights that can have significant implications on applications as well as architectural reconsiderations needed for future larger systems following similar hardware characteristics.

As future work, we plan to apply the insights gained from our studies to specific application kernel cases such as libraries using Cartesian-grid communication (e.g., FFT) which can be impacted by network congestion, or applications that rely on master-worker models (e.g., mpiBLAST) which be impacted from hot-spot communication. We also plan to carry out this study on SiCortex that follows a similar architectural model, but with a Kautz network topology.

References

[1] Berkeley Unified Parallel C (UPC) Project. http://upc.lbl.gov/.

[2] Co-Array Fortran. http://www.co-array.org/.

[3] Global Arrays. http://www.emsl.pnl.gov/docs/global/.

[4] http://www.sicortex.com/products/sc5832.

[5] http://www.research.ibm.com/journal/rd/492/gara.pdf.

[6] InfiniBand Trade Association. http://www.infinibandta.com.

[7] S. Alam, B. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. Vetter, P. Worley, and W. Yu. Early Evaluation of IBM BlueGene/P. In *SC*, 2008.

[8] P. Balaji, A. Chan, R. Thakur, W. Gropp, and E. Lusk. Non-Data-Communication Overheads in MPI: Analysis on Blue Gene/P. In *Euro PVM/MPI Users' Group Meeting*, Dublin, Ireland, 2008.

[9] Overview of the IBM Blue Gene/P project. http://www.research.ibm.com/journal/rd/521/team.pdf.

[10] IBM System Blue Gene Solution: Blue Gene/P Application Development. http://www.redbooks.ibm.com/redbooks/pdfs/sg247287.pdf.

[11] Franck Cappello and Daniel Etiemble. MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 12, Washington, DC, USA, 2000. IEEE Computer Society.

[12] A. Chan, P. Balaji, R. Thakur, W. Gropp, and E. Lusk. Communication Analysis of Parallel 3D FFT for Flat Cartesian Meshes on Large Blue Gene Systems. In *HiPC*, Bangalore, India, 2008.

[13] S. Kumar, G. Dozsa, G. Almasi, D. Chen, M. Giampapa, P. Heidelberger, M. Blocksome, A. Faraj, J. Parker, J. Ratterman, B. Smith, and C. Archer. The Deep Computing Messaging Framework: Generalized Scalable Message Passing on the Blue Gene/P Supercomputer. In *ICS*, 2008.

[14] Naval Research Laboratory. Naval research laboratory layered ocean model (nlom). http://www.navo.hpc.mil/Navigator/Fall99_Feature.html.

[15] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. Performance Comparison of MPI Implementations over InfiniBand Myrinet and Quadrics. In *Supercomputing 2003: The International Conference for High Performance Computing and Communications*, Nov. 2003.

[16] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over InfiniBand with RDMA Support. In *Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS '04)*, April 2004.

[17] J. Liu, J. Wu, S. Kini, R. Noronha, P. Wyckoff, and D. K. Panda. MPI Over InfiniBand: Early Experiences. In *IPDPS*, 2002.

[18] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, March 1994.

[19] MPICH2. http://www.mcs.anl.gov/mpi/mpich2.

[20] Venkatesan Packirisamy and Harish Barathvajasankar. Openmp in multicore architectures. Technical report, University of Minnesota.

[21] Fabrizio Petrini, Wu-Chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.