Understanding Network Saturation Behavior on Large-Scale Blue Gene/P Systems

P. Balaji H. Naik N. Desai

Mathematics and Computer Science,
Argonne National Laboratory,
{balaji, hnaik, desai}@mcs.anl.gov

Abstract

As researchers continue to architect massive-scale systems, it is becoming clear that these systems will utilize a significant amount of shared hardware between processing units. Systems such as the IBM Blue Gene (BG) and Cray XT have started utilizing flat networks (a.k.a. scalable networks) which differ from switched fabrics in that they use a 3D torus or similar topology. This allows the network to only grow linearly with system scale, instead of the super-linear growth needed for full fat-tree switched topologies, but at the cost of increased network sharing between processing nodes. While in many cases a full fat-tree is an over-estimate of the needed bisectional bandwidth, it is not clear whether the other extreme of a flat topology is sufficient to move data around the network efficiently. Thus, In this paper, we study the network behavior of the IBM BG/P using several application communication kernels, and monitor network congestion behavior based on detailed hardware counters. Our studies scale from small systems to up to 8 racks (32768 cores) of BG/P, and show various interesting insights into the network communication characteristics of the system.

1 Introduction

Large-scale systems with hundreds of thousands of cores are available today. As we look forward to even larger scale systems in the future, it is becoming clear that these systems will utilize a large amount of shared hardware. This includes shared caches, shared memory and memory management devices, and shared network infrastructure. Specifically, with respect to the shared network infrastructure, systems such as the IBM Blue Gene (BG) [16, 24] and Cray XT [14, 23] have started utilizing flat networks (a.k.a. scalable networks) which differ from switched fabrics in that they use a 3D torus or similar topology. The primary benefit of using such flat networks is that the number of network components only grows linearly with system size instead of the super-linear growth needed for full fat-tree switched topologies; this means that the network cost and failure rate do not rapidly outgrow the rest of the system.

Though flat networks have their benefits compared to switched fabrics, they come at the cost of increased network sharing between processing nodes. For example, in a 3D torus, each node has six neighbors that it directly connects to. For all other nodes, it has to make multiple hops to reach. Therefore, unless each node communicates with only its physically nearest neighbors, it will be forced to share network links with other communication. Similarly, for systems such as the IBM Blue Gene/P (BG/P) which share a collective network across all nodes, a subset of the nodes using the collective network might mean that another completely independent subset of nodes cannot use it at the same time.

While in many cases a full fat-tree is an over-estimate of the needed bisectional bandwidth, it is not clear whether the other extreme of a flat topology is sufficient to move data around the network efficiently. A full fat-tree topology guarantees that there exists a full set of non-blocking paths between all pairs of nodes in the system. Though, this is not equivalent to saying that all communication in a fat-tree topology is fully non-blocking, the additional network components available in the system do benefit networks from running into congestion issues. A flat network, on the other hand, utilizes significantly lesser number of network components, especially for large systems. So, what does this mean for the data that needs to be communicated? If we reduce the number of network components, the data has to share the available network links, which can potentially lead to more network congestion. At the same time most scalable applications do not communicate with all processes in the system; they mostly rely on clique-based communication which refers to the ability of applications to form small subgroups of processes with a majority of the communication (at least in the performance critical path) happening within these groups. Nearest neighbor (e.g., PDE solvers, molecular dynamics simulations) and cartesian grids (e.g., FFT solvers) are popular examples of such communication [4, 15, 6]. Thus, in such environments would the reduced number of network links in a flat network topology

really increase network congestion?

Thus, in light of these two arguments, it is important for us to understand how such flat networks behave for different application communication patterns. Specifically, are such flat networks sufficient for most common communication patterns? Or are they completely unscalable? How does network congestion behavior vary as we scale up the system size?

In order to answer these questions, in this paper, we perform a detailed analysis of the network congestion on the IBM BG/P system using various application communication kernels. These include global communication between all processes as well as clique based communication mechanisms such as cartesian communication and logical nearest neighbor communication with varying number of dimensions. Our experiments, that use the message passing interface (MPI) [21] as the underlying communication mechanism, study different communication patterns within each communication clique such as all-to-all, broadcast and allgather. Further, for all these cases, we used detailed hardware profiling counters to study network congestion and understand the impact of shared network hardware on such communication patterns. Our experiments that scale from small systems to up to 8 racks (32768 cores) of one of the largest BG/P systems in the world (at Argonne National Laboratory), show various interesting insights into the network communication characteristics of the system.

The remaining part of the paper is organized as follows. We present a brief overview of the IBM BG/P network infrastructure in Section 2. Some prior work done by us that serves as a motivation for the study in this paper is presented in Section 3. Detailed experimental results and the corresponding analysis is presented in Section 4. Other literature related to our work is presented in Section 5, and we draw our final conclusions in Section 6.

2 *Overview of the BG/P Network*

BG/P is the second generation in the IBM BG family. BG/P systems comprise individual racks that can be connected together; each rack contains 1024 four-core nodes, for a total of 4096 cores per rack. Blue Gene systems have a hierarchical structure. Nodes are grouped into midplanes, which contain 512 nodes in an $8 \times 8 \times 8$ structure. Each rack contains 2 such midplanes. Large Blue Gene systems are constructed in multiple rows of racks.

As shown in Figure 1, each node on 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 [11]. Two of them, 10-Gigabit Ethernet and 1-Gigabit Ethernet with JTAG interface[1], are used for file I/O and system management. The other three are used for MPI communication.

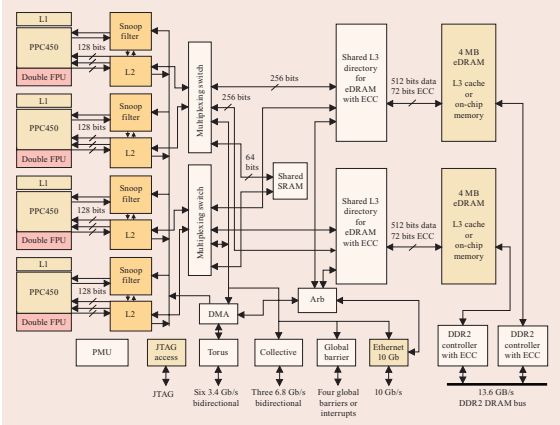

Figure 1: BG/P Architecture [10]

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 (each node has six neighbors). Each link provides a bandwidth of 425 MB/s per direction, for a total bidirectional 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 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 scalable network specifically used for global barriers and interrupts. 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; the DMA engine 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.

The DMA engine on the BG/P maintains a buffer region, known as the DMA FIFO, where it stores data that has been handed over to it by the upper layers, but has not yet

[1] JTAG is the IEEE 1149.1 standard for system diagnosis and management

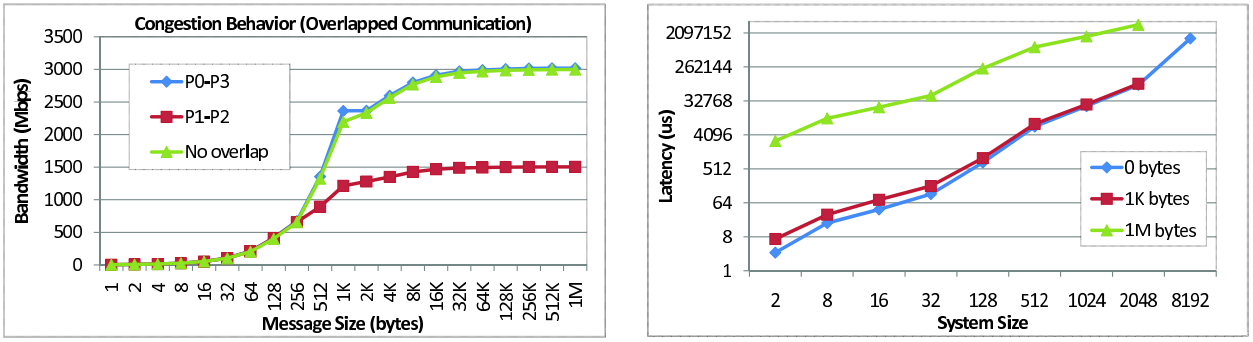

Figure 2: BG/P Network Congestion: (a) Overlapping Communication Performance; (b) Hot-spot Communication Performance

been reliably transmitted on the network. A process can queue data to be sent on the network by adding it to the DMA FIFO. If this FIFO is full, the process can request the hardware for an interrupt when the DMA engine has transmitted some data creating more space in the FIFO. On receiving such an interrupt, the process can refill the FIFO with more data.

3 *Prior Work*

In this section, we present performance measurements from our prior work [9], which motivated the study in this paper.

Figure 2(a) shows the point-to-point bandwidth achieved by two flows that have an overlapping link on the BG/P. 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. 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.

Figure 2(b) shows the hot-spot communication performance, where a single "master" process performs a latency test with a group of "worker" processes, thus forming a communication hot-spot (the graph uses a log-log scale). This test is designed to emulate master-worker kind of communication models. 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.

Figure 3(a) shows the performance of global all-to-all communication. We notice that time taken for `MPI_Alltoall` increases super-linearly, especially for large messages. This represents the worst-case communication behavior, but gives a good indication of the potential congestion issues a flat network might face.

Figure 3(b) shows the impact of different process mappings on the performance of a nearest-neighbor communication kernel, HALO [5]. 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 threefold impact for a system size of 128K processes, because how processes are mapped to the system nodes essentially determines the characteristics of network traffic, and eventually network congestion.

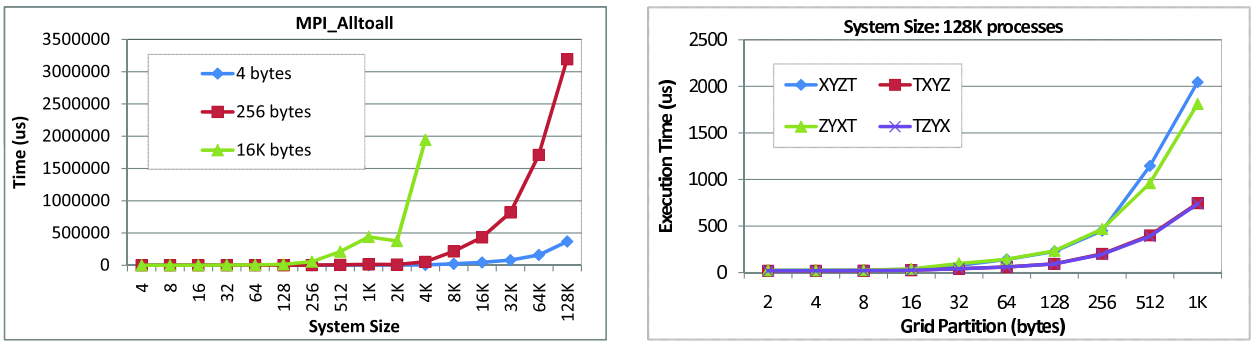

Figure 3: (a) Global All-to-all Communication Performance; (b) Nearest Neighbor Communication

4 *Experiments and Analysis*

As described in Section 3, increasing system sizes have a significant impact on the system communication performance, a lot of which is attributed to network congestion. However, measuring network congestion is not trivial. In this paper, we utilize the BG/P network hardware counters to measure the number of times the node has data to send, but does not have network credits to send data, as a measure of network congestion. While this count does not give the actual amount of *time* for which the host faced congestion, a larger count, in general, means that there is more network congestion.

In this section, we study network congestion behavior for three broad classes of logical communication topologies: (a) global communication (described in Section 4.1), (b) cartesian communication (described in Section 4.2) and (c) nearest neighbor communication (described in Section 4.3). Global communication deals with single communication operations that involve all processes in the system, while cartesian and nearest neighbor deal with smaller cliques, where each process only communicates with a subset of the processes (which is more common for massively scalable applications).

4.1 *Global Communication*

In this section we show the communication impact of global communication with increasing system size.

Alltoall communication: Figure 4 utilizes `MPI_Alltoall`, as a worst-case indication of the network congestion for the system. The legend in the graph represents the number of network stall events noticed along all the dimensions in the 3D Torus. Since this collective only relies on the 3D Torus network, the other networks are not relevant in this case.

For global communication, we notice that the overall network stall count is quite high even for 2048 processes, and the increase in the stall count itself decreases with increasing system size. This behavior is based on the network congestion behavior described in Section 2. Specifically, the process keeps adding messages to the DMA FIFO buffer as long as there is space to add. Once the FIFO buffer is full (because the application is adding messages faster than the DMA engine can send out data), the process just requests the network for an interrupt when there is space available in the DMA FIFO, and gets back to its processing. As the congestion increases, the rate at which the DMA engine can empty its FIFO drops (since the network link is available for data transmission fewer times). Eventually, as the network congestion hits a critical level, as soon as the DMA FIFO advertises the availability of N bytes in the FIFO buffer, the process queues up data corresponding to these N bytes before the DMA engine can transmit any more data. Therefore, when the network reaches this critical congestion level, we do not see any

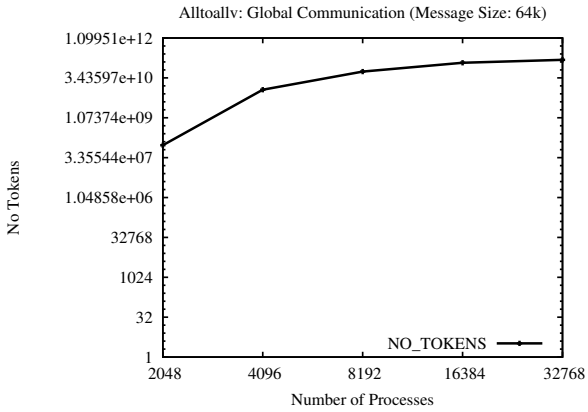

Figure 4: Global Communication (64 KB Message Size): `MPI_Alltoall`

more increase in the network stall counter, as illustrated in Figure 4.

This essentially demonstrates that alltoall kind of communication is fundamentally unscalable on flat torus networks even for smallish system sizes (2048 processes). With systems with hundreds of thousands of processes available today, alltoall communication even within smaller groups of processes can have quite a substantial impact on network congestion and consequently overall communication behavior.

Allgather communication: Figure 5 shows the network congestion behavior of `MPI_Allgather` with increasing system size. For large messages, `MPI_Allgather` utilizes the collective network for its communication. Thus, in this experiment, we measure the number of times the end node has data to send, but the network is busy with other communication.

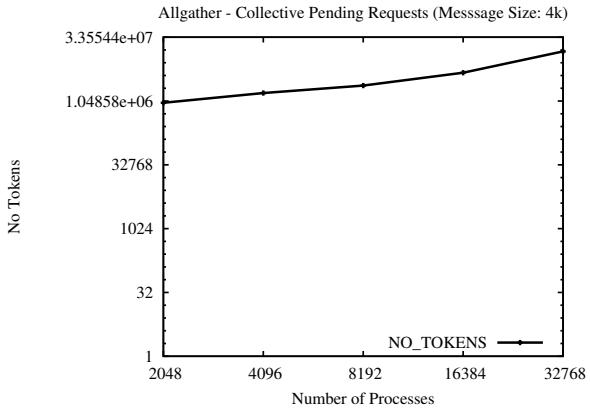

Figure 5: Global Communication (4 KB Message Size): `MPI_Allgather`

For medium-to-large sized messages, `MPI_Allgather` is implemented using multiple `MPI_Bcast` messages, one from each process. However, since there is only one shared collective network, all these broadcasts have to wait for the network to become available. Thus, with increasing system size, the number of stalls each process sees increases as well. In Figure 5, we see a similar trend as `MPI_Alltoall` even for `MPI_Allgather`; that is, the overall network stall count is quite high even for 2048 processes, and stays relatively unchanged with increasing system size. This is attributed to a similar reason as `MPI_Alltoall`.

Broadcast communication: We have also done measurements with `MPI_Bcast`. But that does not show any network stalls, since the collective network is only used once within each broadcast operation, unlike an allgather operation where each node does a broadcast on the same network. Thus, there would not be any congestion. These results are not shown in this paper.

4.2 *Cartesian Communication*

In this section we measure the network congestion behavior of BG/P for cartesian communication patterns. Specifically, the application logically lays out all the processes in the system into an N-dimensional grid, and communicates only with processes in one of the dimensions at a time. For example, in a 2-dimensional grid, a process communicates only with the other processes in its row or its column.

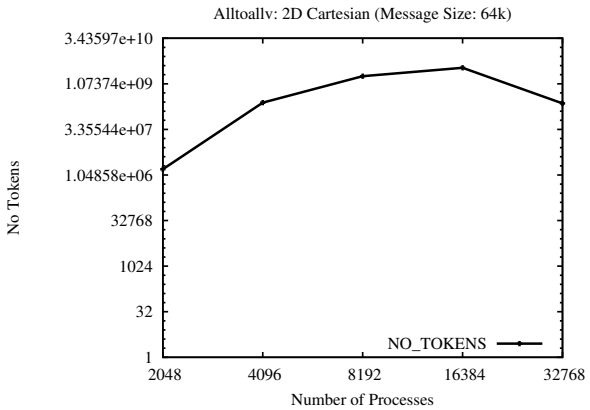

Figure 6: Cartesian Communication (64 KB Message Size): `MPI_Alltoall`

Figure 6 shows the network congestion behavior for a 3D Cartesian grid where each process communicates with other processes on a 2D plane (which is a subset of the processes in the system). However, as shown in the figure, the network congestion behavior for this case is not very different from that of global communication, as illustrated in Figure 4.

To understand this behavior, we first need to look at the correspondence between the physical topology of the processes and the logical cartesian grid that the application forms using a call to `MPI_Dims_create`. As shown in Table 1, the processes in the system are laid out across four physical dimensions—three of these (X, Y, Z) correspond to the 3D torus and the fourth (T) is the cores on each node (so it can go to a maximum of 4). The logical dimensions, on the other hand, are as returned by `MPI_Dims_create`.

With this layout, a 2D plane in the logical 3D process grid

Processes	Nodes	Physical Dimensions (XYZT)	Logical Dimensions (XYZ)
2048	512	8×8×8×4	32×8×8
4096	1024	8×8×16×4	64×8×8
8192	2048	8×8×32×4	128×8×8
16384	4096	8×16×32×4	128×16×8
32768	8192	8×32×32×4	128×32×8

Table 1: Summary of partition shapes

would correspond to a large number of nodes in the system. For example, consider the 2048-process case; here, a logical 2D plane using X and Y dimensions would have 32×8 (256) processes. However, since the physical topology has processes laid out as 8×8×8×4, these groups of 256 processes would correspond to segments of size 8×8×4 in the physical topology. Thus, each core on a node (T dimension) would belong to a different logical 2D plane, causing traffic in one logical 2D plane to interfere with the traffic in the other logical 2D planes.

This kind of mismatch between the logical process layout and the physical layout results in traffic interference and network congestion, even though each process is "logically" communicating within its clique of processes.

4.3 *Nearest Neighbor Communication*

In this section we study the network congestion behavior when the application has a nearest-neighbor kind of communication pattern. In such communication, the processes are laid out as an N-dimensional logical grid (in a same way as a cartesian grid), but each process communicates only with its neighbors in each dimension (unlike a cartesian grid where a process communicates with all other processes in each dimension).

Such communication is typically done using one of two methods. In the first method, each process does `MPI_Alltoallv` but specifies a zero data count to all processes other than its neighbors (for example, the PETSc [6] numerical library has an option to perform this kind of communication). In the second method, each process manually performs point-to-point communication with its logical neighbors.

Figures 7 and 8 the performance of both methods for 2D and 3D logical process grids. As expected, we notice no significant difference between the two methods. For 2D process grids in both cases, however, we notice that there is typically no significant congestion, except for some cases (e.g., 8192 processes), depending on the physical topology of the partition. However, for 3D process grids, we notice that the congestion is significantly higher. This is because, as described in Section 4.2, there is no tight correspondence between the physical topology and the logical topology, especially for higher dimensional process grids. Thus, the "nearest neighbors" for a process in a "logical" process grid, can be anywhere in the physical layout, leading to significant requirements on the network bisectional bandwidth even in this case.

5 *Related Work*

There has been previous work on understanding the communication and non-communication overheads (in the context of MPI) on various architectures [18, 17, 19, 13, 8, 7]. However, none of this work looks at the network saturation behavior that is becoming increasingly important with system size, which is the focus of this paper.

There has also been work recently to understand whether MPI would scale to such massively large systems, or if alternative programming models are needed. This includes work in extending MPI itself [20] as well as other models including UPC [1], Co-Array Fortran [2], Global Arrays [3], OpenMP [22] and hybrid programming models (MPI + OpenMP [12], MPI + UPC). While this paper utilizes MPI as a tool for measuring the network congestion behavior, most of the insights are independent of MPI, and do give a general indication of potential pitfalls other models might run into as well.

6 *Conclusions and Future Work*

In this paper, we performed a detailed analysis of the congestion behavior on the IBM Blue Gene/P system, in order to understand the impact of increasing system scales on different application communication patterns. We studied various application communication kernels, including global communication between all processes as well as clique based communication mechanisms such as cartesian communication and logical nearest neighbor communication with varying number of dimensions and different communication patterns. Our experiments that scale from small systems to up to 8 racks (32768 cores) of BG/P, show various interesting insights into the network communication characteristics of the system.

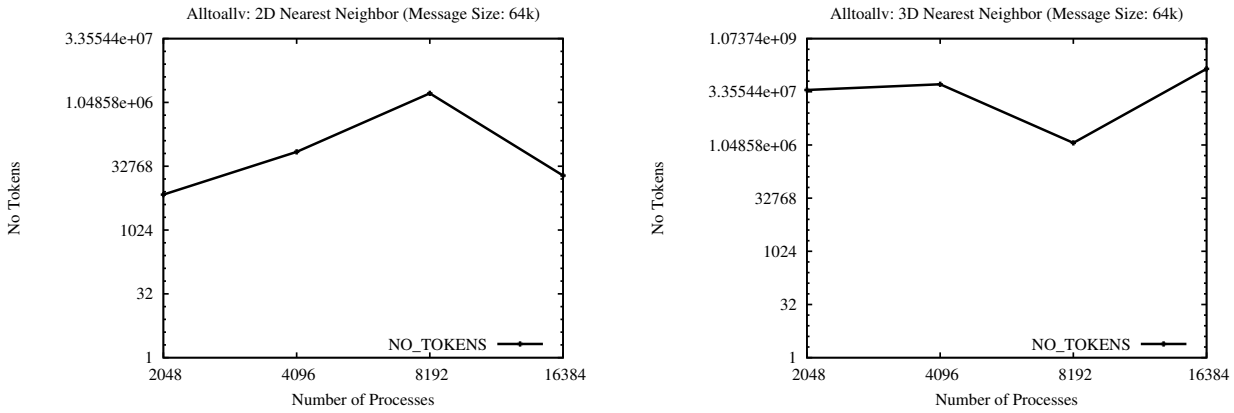

Figure 7: Nearest Neighbor Communication: `MPI_Alltoallv`: (a) 2D Process Grid; (b) 3D Process Grid

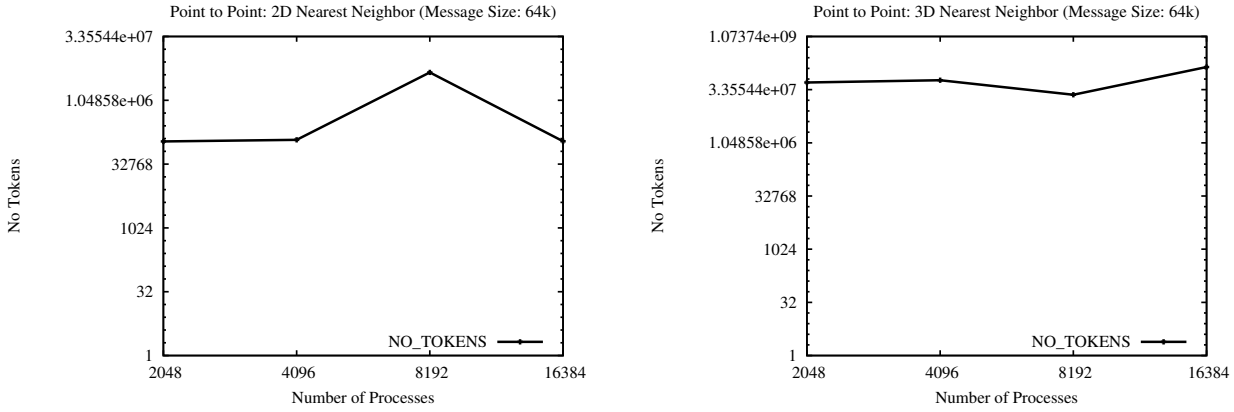

Figure 8: Nearest Neighbor Communication: point-to-point: (a) 2D Process Grid; (b) 3D Process Grid

Different process mappings have different network congestion behavior resulting in different performance. As future work, we plan to investigate the magnitude of this impact, and ways in which the best mapping can be predecided.

7 Acknowledgment

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.

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] GROMACS. http://www.gromacs.org/.

[5] Naval Research Laboratory Layered Ocean Model (NLOM). http://www.navo.hpc.mil/Navigator/Fall99_Feature.html.

[6] PETSc. http://www-unix.mcs.anl.gov/petsc/.

[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; **Best Paper Award***, Dublin, Ireland, 2008.

[9] P. Balaji, A. Chan, R. Thakur, W. Gropp, and E. Lusk. Toward Message Passing for a Million Processes: Characterizing MPI on a Massive Scale Blue Gene/P. In *the Proceedings of the International Supercomputing Conference (ISC); **Best Paper Award***, Hamburg, Germany, June 2009.

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

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

[12] 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.

[13] 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.

[14] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[15] M. Frigo and S.G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 2005.

[16] Alan Gara, Matthias A. Blumrich, Dong Chen, George L.-T. Chiu, Paul Coteus, Mark Giampapa, Ruud A. Haring, Philip Heidelberger, Dirk Hoenicke, Gerard V. Kopcsay, Thomas A. Liebsch, Martin Ohmacht, Burkhard D. Steinmacher-Burow, Todd Takken, and Pavlos Vranas. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2-3):195–212, 2005.

[17] 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. Int'l Parallel and Distributed Processing Symposium (IPDPS 04), April 2004.

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

[19] Jiuxing Liu, Balasubramanian Chandrasekaran, Jiesheng Wuand Weihang Jiang, Sushmitha Kini, Weikuan Yu, Darius Buntinas, Peter Wyckoff, and Dhabaleswar 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.

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

[21] Message Passing Interface Forum. MPI-2: A Message Passing Interface Standard. *High Performance Computing Applications*, 12(1–2):1–299, 1998.

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

[23] S. L. Scott and G. M. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Proceedings of the Symposium on High Performance Interconnects (Hot Interconnects 4)*, pages 147–156, August 1996.

[24] IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 52(1-2):199–220, 2008.