Implementing Optimized Collective Communication Routines on the IBM BlueGene/L Supercomputer

Sam Miller
samm@scl.ameslab.gov
Computer Science 425
Prof. Ricky Kendall
Iowa State University
Spring 2005

Abstract. BlueGene/L is a massively parallel supercomputer that is currently the fastest in the world. Implementing MPI, and especially fast collective communication operations can be challenging on such an architecture. In this paper, I will present optimized implementations of MPI collective algorithms on the BlueGene/L supercomputer and show performance results compared to the default MPICH2 algorithms on a sample 4096 node BlueGene/L computer installed at IBM's Rochester, MN facility.

1. Introduction

The IBM BlueGene/L supercomputer represents a performance increase of an order of magnitude over the previous fastest supercomputer in the world [1]. With an estimated peak performance of 180 or 360 Teraflops depending on the configuration mode, and 65,536 nodes, it also represents a scalability jump of nearly two orders of magnitude over previous large scale parallel systems [2]. There are several challenges when implementing a standards compliant MPI library on such a system. In this paper, I will review the design choices made by IBM engineers responsible for implementing MPI and optimizing collective communication on the BlueGene/L. Section 2 will provide a brief introduction to the BlueGene/L hardware. Section 3 will introduce the communication networks present in each node. Sections 4 and 5 will discuss the network and system software. Section 6 will present MPICH, the MPI implementation chosen for BlueGene/L. Section 7 will discuss general techniques for optimizing collective routines on BlueGene/L. Sections 8 and 9 will discuss the optimized collective algorithms on the torus and tree networks. Lastly, section 10 will present performance results of the optimized BlueGene/L collective communication algorithms and the MPICH2 algorithms.

2. Node Hardware

The BlueGene/L supercomputer is a deviation from the recent trend in large parallel systems designed by clustering several SMP nodes together with very fast interconnection networks. Instead of using the fastest available processor, IBM chose to use a readily available PowerPC 440 processor combined with system-on-a-chip technology to integrate floating-point units, caches, a DDR memory controller, and the logic for five different interconnection networks together on one ASIC. The PowerPC 440 was developed for embedded applications such as

digital media set top boxes or network routers. Thus, the PowerPC 440 exhibits extremely low power consumption at a modest clock rate (700 Mhz) in addition to a low cost.

Each BlueGene/L compute node consists of a single ASIC containing two 32-bit PowerPC 440 cores. Each core contains two custom 64-bit floating-point units that can operate in parallel, each contains two 32 by 64 bit register files for load and store operations. The floating point units are capable of dispatching two fused multiply-adds per cycle. Therefore, at the target 700 Mhz clock rate, the performance will be 2.8 GFlops per node or 5.6 GFlops per node when both processors are used. This is how the theoretical peak of 180 or 360 GFlops is achieved in the full 65,536 node configuration.

To build the entire BlueGene/L system, two ASICs along with 256MB or 512MB of DDR memory are combined to form a compute card. Sixteen compute cards are put together in a node board; thirty-two node boards are put together into a cabinet for a total of 65,536 nodes and 16 or 32 terabytes of memory. Each compute node does not contain tertiary disk storage, instead special nodes are designated as I/O nodes when the system is initially setup. More information about the I/O nodes will be presented in section 3. Figure 1 shows how the entire 64 rack system is built.

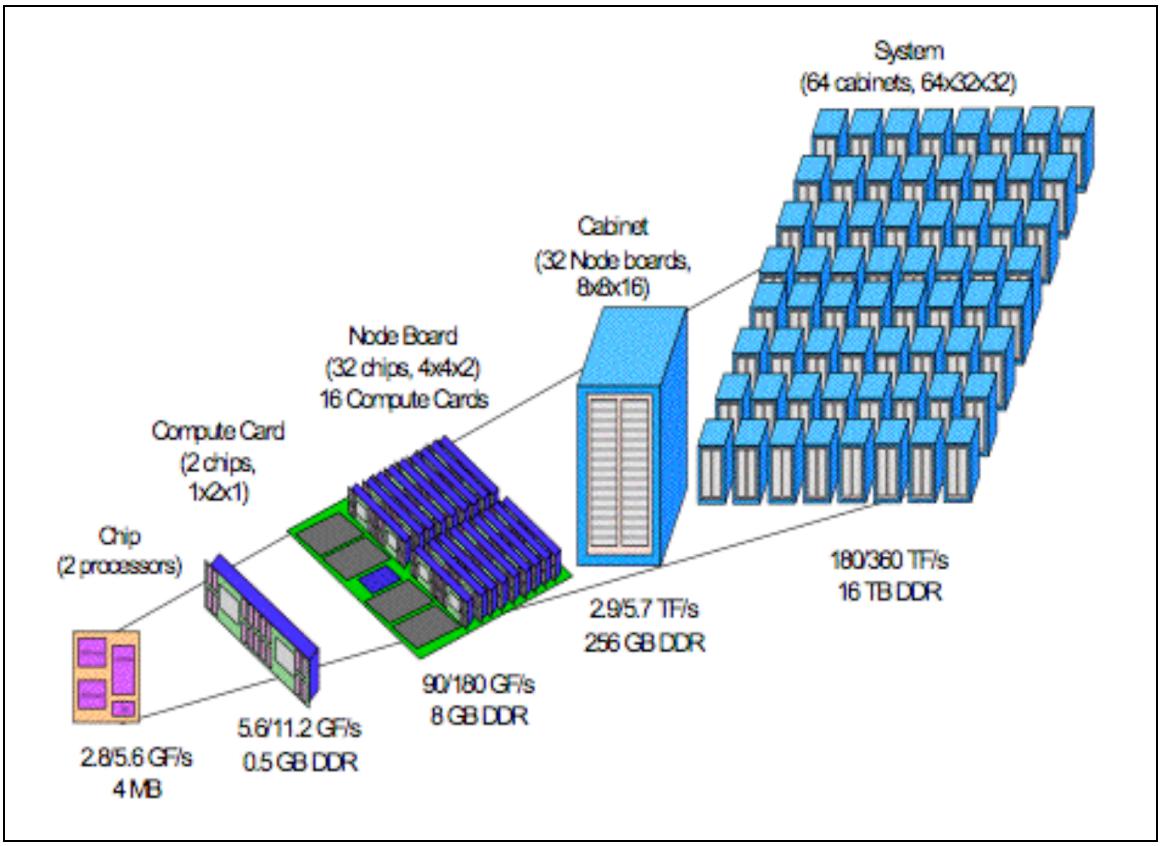

Figure 1. BlueGene/L System Overview. Taken from [10].

It is important to realize the physical dimensions of this system are significantly smaller than its predecessors. Whereas previous supercomputers often require large facilities to operate, the

65,536 BlueGene/L installation at Lawrence Livermore National Laboratories will occupy less space than a typical tennis court, or about 2,500 square feet [10].

3. Communication Networks

Each BlueGene/L node has five different networks. The primary network for communication between nodes is the three-dimensional torus network. This is also the primary network for the MPI library. A torus network was chosen because it provides high bandwidth nearest neighbor connectivity, which is common in many scientific applications [6]. Such a network has also been used successfully in other supercomputers, such as the Cray T3E. Each compute node is connected to its 6 neighbors through bi-directional links with 154 Mbytes/s payload bandwidth in each of those directions [3]. The full 64 rack system will form a 64x32x32 three-dimensional torus. The underlying network hardware of the torus network guarantees reliable, deadlock free delivery of variable length packets [3].

The tree network encompasses all of the nodes and is used by MPI for some collective operations, file I/O operations, and by the system software to facilitate program loading on each of the nodes. The tree network payload bandwidth is 337 Mbytes/s [3]. The I/O nodes communicate with the compute nodes using the tree network. The three remaining networks are the Gigabit Ethernet network, JTAG network, and the global interrupt network. The Gigabit Ethernet network only connects a small number of nodes, typically only the I/O nodes. The JTAG network is used for node booting and some control functionality. Lastly, the global interrupt network is used for job startup, checkpointing, and barrier operations.

4. System Software

Each compute node in the BlueGene/L system runs a very small and lightweight compute node kernel (CNK) that is written in approximately 5000 lines of C++ code [3]. The CNK provides no more and no less than two threads, each of them running on one PowerPC 440 core. The CNK provides a standard glibc interface, thus enabling many scientific applications to be ported easily.

The I/O nodes run a standard PowerPC Linux operating system instead of a stripped down kernel like the compute nodes. Some of the CNK system calls are not directly executed on the compute node. For example, what should a fwrite system call do when the compute node has no disk? In this case, the CNK sends a message to its I/O node using the tree network. Each I/O node runs a daemon called the console I/O daemon (ciod). When the ciod receives the fwrite tree packets, it will perform the system call on the mounted file system and returns a status message to the original CNK. In the current configuration, each I/O node is responsible for 64 compute nodes. Other configurations are possible if heavy I/O bandwidth is anticipated. The collection of one I/O node and its associated compute nodes is called a processing set, or pset [3].

5. Network Software

Users interact with the networks through the three communication software layers. The topmost layer is the MPI library. The middle layer is the Message Layer, and the bottom layer is the

Packet Layer. The Packet Layer does very little aside from initializing the network hardware and facilitating the sending and receiving of network packets. It is a simple stateless interface to the torus, tree, and global interrupt network hardware. The Message Layer is an active message system that sits on top of the Packet Layer. It allows the transmission of arbitrary buffer sizes among the compute nodes. It consists of a point-topoint component, a collective communication component, and a component for debugging. The internal structure of the Message Layer uses the equivalent of MPI_COMM_WORlD ranks to address its neighbors. It translates these ranks into x, y, and z coordinates in the torus network, which are used by the Packet Layer to route the messages to their destination.

Several protocols exist in the message layer for sending point-to-point messages. The reason for multiple protocols is due to the performance benefits of multiple routing strategies when applied to various message lengths. The one-packet protocol handles short messages that can fit into one packet. These messages are sent using deterministic routing, which routes packets along the same x, y, and z path, to avoid out of order issues. The eager protocol uses deterministic routing as well, and handles messages between 200 bytes and 10 kbytes. Its purpose is to maximize network bandwidth. The rendezvous protocol is necessary because the two previous protocols both use deterministic routing, which result in reduced network efficiency compared to adaptive routing. However, with adaptive routing, packet delivery order is not guaranteed which can cause problems for reorganizing packets on the receiving end. To mitigate this problem, the rendezvous protocol sends an initial scout packet via deterministic routing to the source node. The scout packet is required to ask permission from the receiver to send data, as well as to establish the destination address. Now, each rendezvous protocol packet can carry its own destination address and can be considered "self-contained." Thus, the adaptive routing algorithm is more appropriate here since it will maximize bandwidth. The initial scout packet handshake costs 1500 cycles on both the sender and receiver [3]. However, the cost of handling a rendezvous packet on the receiving end is reduced by 150 cycles over the eager protocol since the destination address is self-contained. This reduces the performance impact of the initiate handshake and makes the rendezvous protocol suitable for messages greater than 5 kbytes in length [3].

Sending point-to-point messages incurs a relatively high overhead in terms of CPU cycles. For the remainder of this paper, I will present the optimized collective routines and present some performance results showing why they are superior to the default MPICH2 collective algorithms using point-to-point communication.

6. MPICH

Due to its highly distributed nature, MPI is the natural parallel programming model of choice for the BlueGene/L system. Most MPI implementations would have extreme scalability issues with the large number of MPI processes present in the BlueGene/L system. The starting point for an MPI library for the BlueGene/L system was MPICH2, developed by Argonne National Laboratory. It is an open source, freely available, MPI-1 compliant implementation of the Message Passing Interface standard. MPICH2 uses an Abstract Device Interface allowing implementers to simplify the job of porting it to different architectures [3]. The ADI layer is

described in terms of MPI requests and various functions to manipulate these requests. The BlueGene/L implementation of the ADI layer turns these requests into Message Layer messages.

7. MPI Collective Algorithms

Implementing a standards compliant MPI library was an early priority for the BlueGene/L team. The reason for this is simple, once point-to-point messages have been implemented, the standard collective communication algorithms in MPICH2 will function correctly. However, they may not perform as well as the hardware specifications may allow. The performance of MPI collectives is critical to overall application performance in many parallel programs. It is very rare to find an embarrassingly parallel type application, which utilizes few if any collective routines. A 5-year study by Rolf Rabenseifner at the University of Stuggart revealed that more than 40% of the time spent in MPI functions was in MPI_Reduce and MPI_Allreduce [9]. Application performance could be substantially improved by optimizing only these two collective routines, and further improved by optimizing other commonly used collectives. In this section I will briefly present strategies for optimizing collective algorithms, and some of the approaches taken by the IBM BlueGene/L engineers. In sections 8 and 9, I will describe the collective communication algorithms used by MPICH2, and expand on the optimized BlueGene/L algorithms for the following collective routines: broadcast, allreduce, reduce, barrier, and alltoall.

The default collective operations in MPICH2 are extremely poor performing on the BlueGene/L system due to a number of factors. Foremost, the MPICH2 default implementations of collective operations assume a crossbar type network. This would more often than not suffer from poor mapping when using the specialized 3D torus network. Secondly, point-to-point messages incur relatively high overhead in terms of CPU cycles. Collective operations with short message sizes would most certainly suffer greatly from this overhead. Lastly, a few important features of the network hardware are hidden when using point-to-point messages, such as the deposit bit feature of the torus network messages, or the built-in ALU of the tree network.

In general, optimizing MPI collective routines is not as straightforward as it may seem. There are several factors that determine the optimization algorithm for the appropriate collective routine. Typically the message size and the communicator shape are the primary factors. Optimizing bandwidth utilization for larger messages and optimizing latency for smaller message is the primary goal for the BlueGene/L implementation [8]. It should also be noted that in the event an optimized collective routine is not available for a certain communicator configuration and/or message size, the default MPICH2 collective routine is used. The idea behind optimizing collective routines for the BlueGene/L system is not to create better general-purpose algorithms than what already exist, but to create optimized algorithms that are used when the conditions are right.

The conditions for selecting the appropriate algorithm are made locally by each node after the collective routine has been invoked. This creates a problem of a potential deadlock if the user invokes a collective with different sized buffers across the participating communicator. While this is illegal according to the MPI specification, some legal calls such as using heterogeneous data types could cause similar issues [4]. For this reason, certain algorithm decisions are made

globally across the participating communicator before executing the algorithm. This is accomplished by using another collective, typically MPI_Allreduce with a one packet message. This results in an increase of latency for the optimized collective, and thus this method is only used when the increase in latency can be offset by bandwidth gains in collective routines with long message sizes [8].

In the following two sections I will describe the optimized algorithms for broadcast, barrier, reduce, allreduce, and alltoall on both the torus and tree networks and compare them to the default algorithms present in MPICH2. I will concentrate on the algorithms optimized for long message sizes since the current published research by the BlueGene/L team for optimized collective routines with short message sizes does not perform much better than the default MPICH2 collectives in most cases [8].

8. Torus Collective Routines

The optimized collective routines on the torus network all exploit two common features of the BlueGene/L hardware. First, they all require a rectangular communicator shape. Upon partition creation, MPI_COMM_WORLD is always rectangular for that particular BlueGene/L partition. However, sub communicators may or may not maintain this rectangular property. The rectangular communicator is important because all of the torus optimized collective routines use the deposit bit feature of the torus network where each message is deposited on every node they touch as it traverses the link towards its destination. Using this network feature would not be possible with a non-rectangular communicator size.

8.1 Broadcast

A broadcast collective operation is the simplest collective operation to understand. In this operation, a process called the root process has data that needs to be sent to all the other processes. There are two default broadcast algorithm in MPICH2 depending on the message size. For small messages, it uses a binomial tree algorithm where the root process sends its data to process (root + p/2). Both processes then act as the root process in their own subtree and the algorithm continues recursively. For large messages it performs a scatter then an allgather. In either case, these two algorithms perform poorly on BlueGene/L due to their inability to recognize the special torus topology and their high CPU overhead [8]. For an n-dimensional mesh, the BlueGene/L optimized implementation of a broadcast consists of n concurrently executing stages where 1/n pieces of the message are sent in each n dimension per stage. The optimized broadcast algorithm exploits the deposit bit feature of the torus network where all the nodes receive each message as it traverses towards its destination. Figure 2 shows a graphical representation of this algorithm on a 2 dimensional mesh topology. The black circle represents the root node of the broadcast. The red lines represent the one half of the message traversing on the torus network and the blue lines represent the other half. This algorithm features low CPU overhead on the nodes that receive packets with the deposit bit turned on. The only nodes who have to re-inject packets onto the network are those who turn the message by 90 degrees in Figure 2.

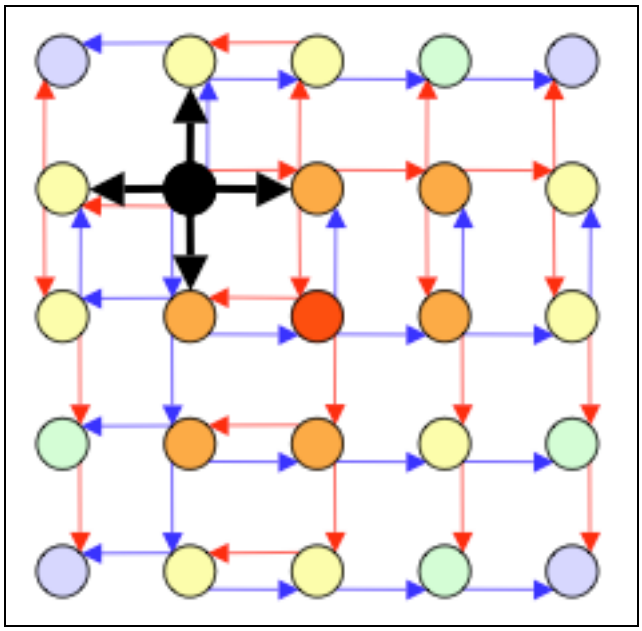

Figure 2. BGL torus optimized broadcast on a 2D mesh topology.

8.2 Reduce and Allreduce

A reduction collective routine is very similar to the opposite of a broadcast. In this case, each node has a piece of data that needs to be reduced to the root processor. The reduction operation can be as simple as addition or maximization, or as complex as a user defined associative operation. The default reduction algorithm in MPICH2 uses a binomial tree for short messages, and a reduce scatter, followed by a gather to the root process for longer messages [9]. The BlueGene/L optimized algorithm could use the same ideas as the broadcast algorithm, but in reverse. There is one important difference though, the reduction algorithm cannot use the deposit bit feature because each node must apply the specified reduction operation before sending its result. The busiest nodes will be those that have to combine incoming data from multiple neighbors before sending their result. Therefore, to minimize the number of incoming data sources, a Hamiltonian path chaining all the nodes in the communicator together will ensure each node receives the data from one neighbor [8]. Figure 3 shows a sample Hamiltonian path for a 4x4x4 3D mesh. Clearly this algorithm optimizes bandwidth at a severe cost to latency. In section 9 I will describe optimized reduce and allreduce algorithms using the tree network that is better performing than this torus network algorithm. An optimized allreduce algorithm is very similar to the reduce algorithm except it initiates a broadcast after the completion of the reduction.

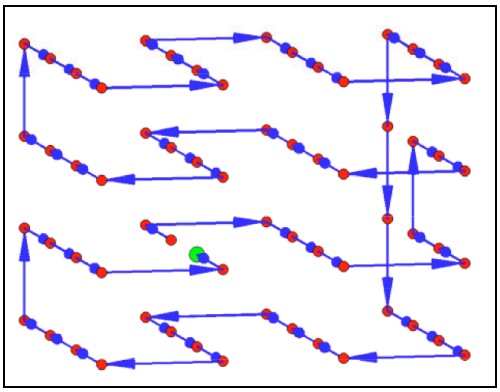

Figure 3. Hamiltonian path for optimized reduction collective algorithm.

8.3 Alltoall and Alltoallv

The alltoall collective routine is an operation where each process has unique data to be sent to every other process. Optimizing alltoall and its vector variant can be extremely tricky. This is evident by the fact that MPICH2 has four separate algorithms for implementing the alltoall collective. None of these algorithms perform particularly well on the BlueGene/L system, mostly due to high CPU overhead, and creation of hot spots within the torus network [8]. The BlueGene/L optimized alltoall algorithm does not use the deposit bit feature nor does it require a rectangular communicator size as the previous two torus collective algorithms have required. The optimized algorithm keeps CPU overhead low by not utilizing point-to-point messages, and it avoids the network hot spots experienced by the default algorithms in MPICH2 by using randomized packet injection in the torus network [8]. The randomization is accomplished by creating a list of MPI ranks, then scanning through the list to pick a destination for the next packet. Each node uses the same random number seed, yet they are all offset into the list differently to minimize potential conflicts in the network. The randomized packet injection can potentially cause a problem on the memory subsystem. Since the payload of each torus packet is up to 240 bytes, or 8 cache lines, randomly switching between destinations for packets will not efficiently use the cache prefetching engine. Therefore, the optimized algorithm sends multiple packets from adjacent cache lines to each destination before advancing to the next entry in the list [8]. Sending too many packets per destination can create network hot spots similar to what is seen with the default MPICH2 algorithm, therefore the IBM engineers determined that two packets per destination was the optimal trade-off in network bandwidth and cache performance.

The alltoallv collective routine uses naturally unbalanced communication due to its implied meaning. The only difference in this algorithm compared to the normal alltoall algorithm is the destination list may contain ranks that no longer require data to be sent to them. To avoid the unnecessary CPU overhead caused by scanning empty slots in list, the list is rearranged in place after a node has no more data to be received.

9. Tree and Global Interrupt Collective Routines

Using the tree network for the allreduce, reduce, and broadcast collectives is a natural choice due to the tree network's design. Since the tree network has a built in arithmetic unit, performing the reduction operations on each node is trivial. The broadcast operation can be performed by idling the arithmetic unit since it is not needed in that case. In either case, the logical root of the reduction or broadcast operation sends the message up the tree to the physical root of the tree, which then sends the message down the tree. Because the arithmetic unit present in the tree hardware can only perform fixed point arithmetic, doing a reduction operation with floating point numbers is much more CPU intensive due to the need to implement floating point arithmetic in software. In section 10 I will show the current implementation of a reduce or allreduce collective using floating point numbers performs much better on the torus network than the tree network. In the current implementation, the optimized tree collective routines are limited to the MPI_COMM_WORlD communicator only, and they do not work at all for machine configurations below 32 nodes [8].

Implementing a barrier collective using the global interrupt network is an obvious choice due to its extremely low latency of 1.5 microseconds for the full 65,536 BlueGene/L node configuration. The only disadvantage of the global interrupt network is that it can only be used on communicators of 32, 128, 512, or multiples of 512 nodes. For other sized communicators, the default MPICH2 implementation can be used with some obvious performance penalties.

10. Performance Results

The bandwidth performance numbers in this section are obtained from debugging benchmarks in the BlueGene/L Message Layer. Table 1 shows the sample machine sizes used and their respective torus dimensions. A "T" next to the torus dimensions indicates a true torus configuration. Configurations below 512 nodes are unable to be booted in a torus configuration and are thus limited to a 2D mesh.

Figures 4 through 9 are the same figures present in [8]. Figure 4 and Figure 5 show the bandwidth performance of the BlueGene/L optimized allreduce algorithm on 512 nodes when sending various sizes integer and double precision messages. When sending integer arguments, it is clear that the tree algorithm performs significantly better than the torus or default MPICH2 implementation. In this case, the theoretical peak bandwidth for the tree (337 Mbytes/sec) is almost achieved in Figure 4. When sending double precision arguments, the Hamiltonian path torus algorithm performs better as the message size increases. The tree algorithm is poor performing for all message sizes using floating point numbers since the tree's built in arithmetic unit is unable to perform the floating point arithmetic.

Figure 6 shows the performance of the optimized BlueGene/L broadcast implementation for both the torus and tree networks outperforming the default MPICH2 implementation. It is clear that for large message sizes, both the torus and tree broadcast algorithms are superior to the MPICH2 implementation. Recall however, that the tree algorithm can only be used in the MPI_COMM_WORLD communicator; the torus algorithm can be used in any rectangular communicator.

Figure 7, Figure 8, and Figure 9 show the performance of the optimized BlueGene/L alltoall implementation for the torus network in configurations of 32, 512, and 4096 nodes. Notice that the near peak bandwidth is achieved in the 512 node case. This tends to show that this algorithm is better performing for cubic topologies. In all cases however, the optimized algorithm outperforms the MPICH2 algorithms.

Table 1. Torus topologies on different machine sizes.

Number of Nodes	Torus Dimensions
32	4x4x2
64	8x8x4
128	8x4x4
256	8x8x4
512	8x8x8 (T)
1024	8x8x16 (T)
2048	8x16x16 (T)
4096	8x32x16 (T)

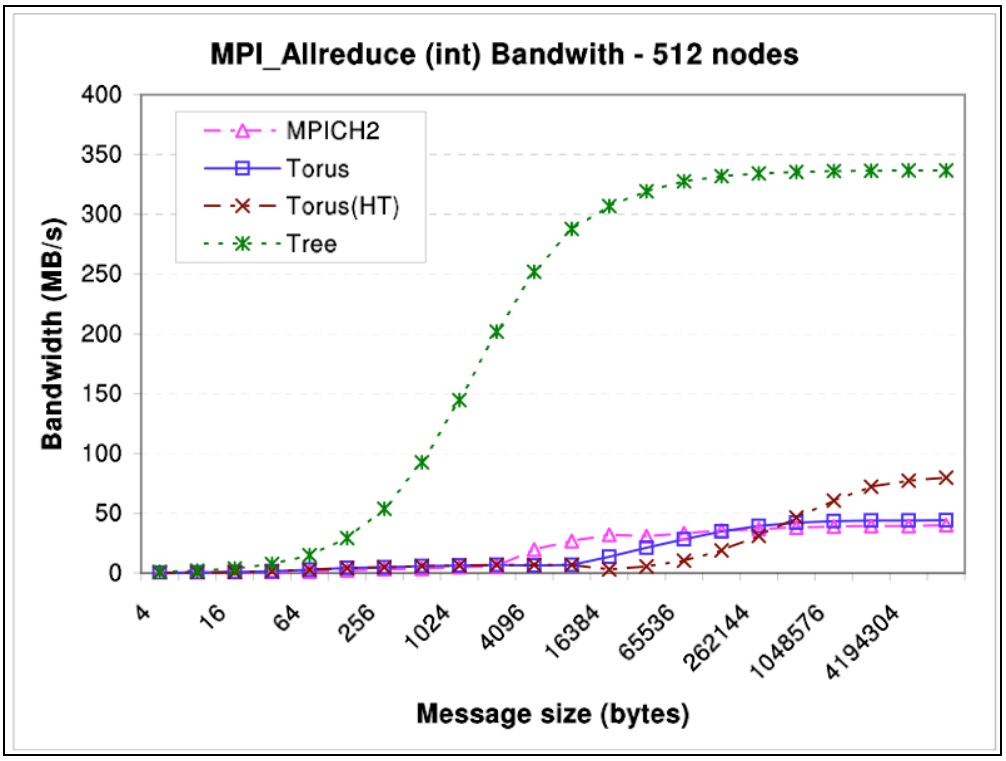

Figure 4. MPI_Allreduce performance when sending various sized integer arguments.

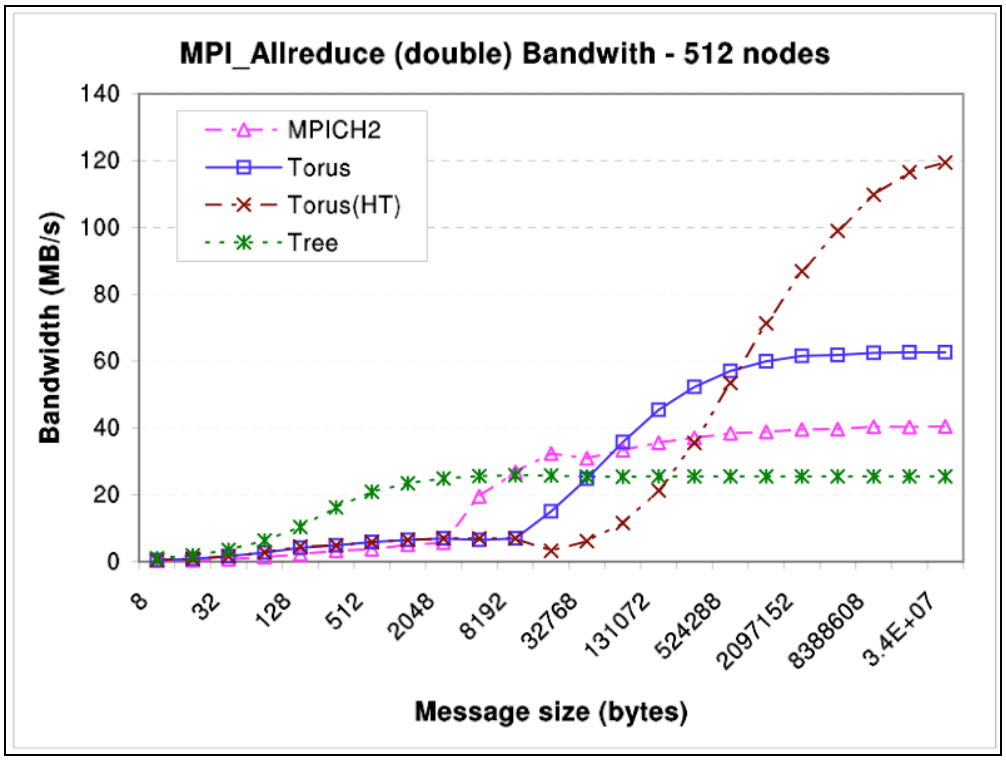

Figure 5. MPI_Allreduce performance when sending various sized double precision arguments.

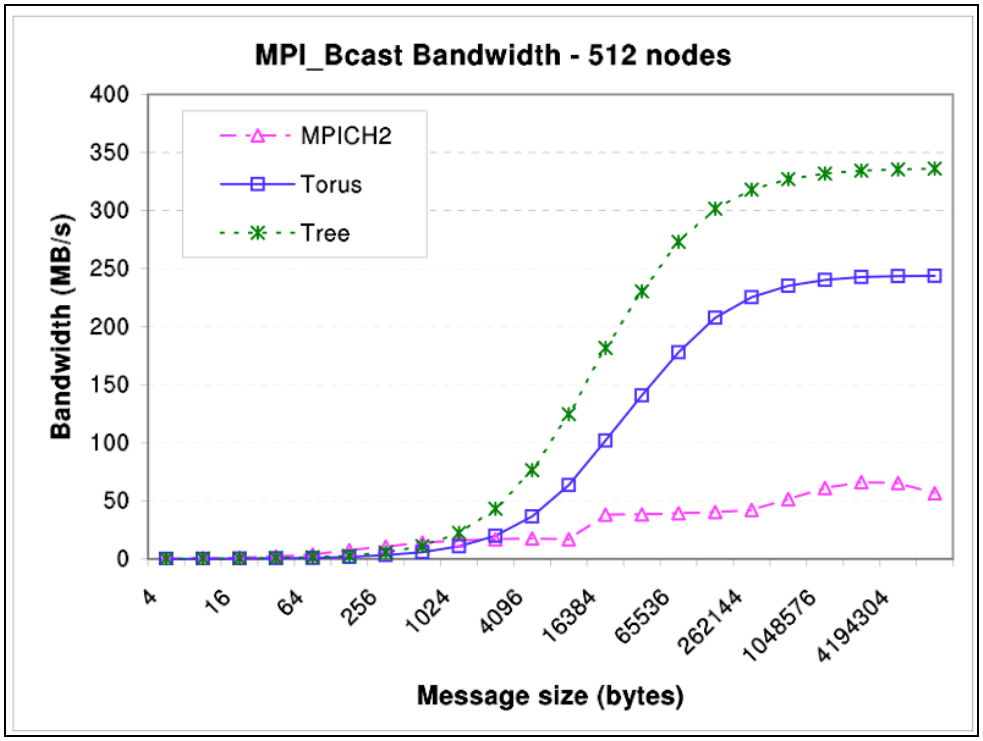

Figure 6. MPI_Bcast performance when sending various sized arguments.

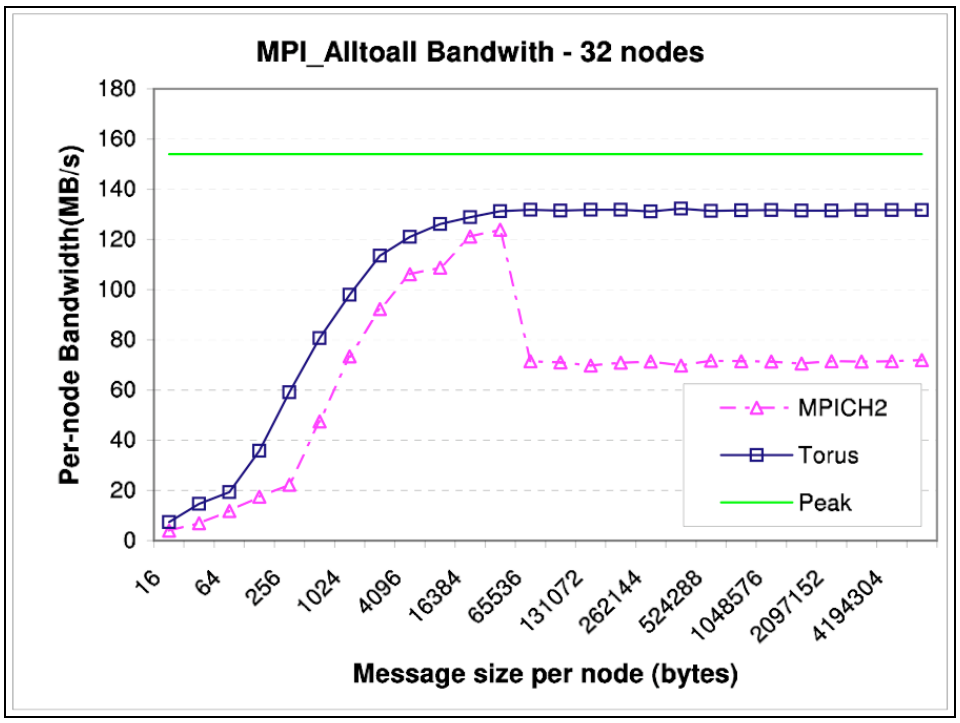

Figure 7. MPI_Alltoall performance on 32 nodes when sending various sized messages.

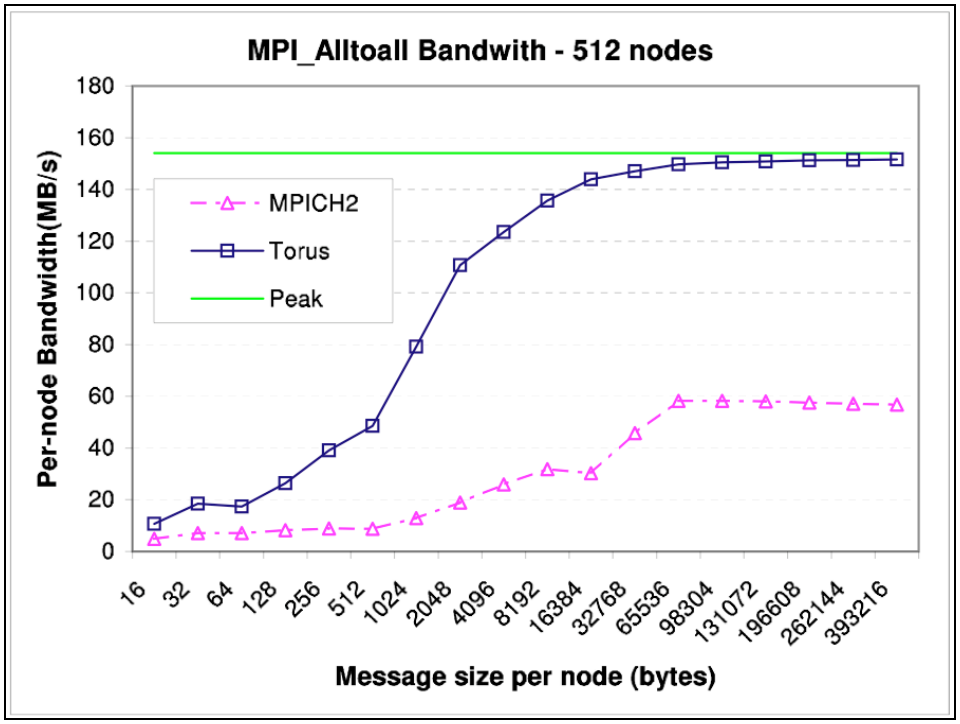

Figure 8. MPI_Alltoall performance on 512 nodes when sending various sized messages.

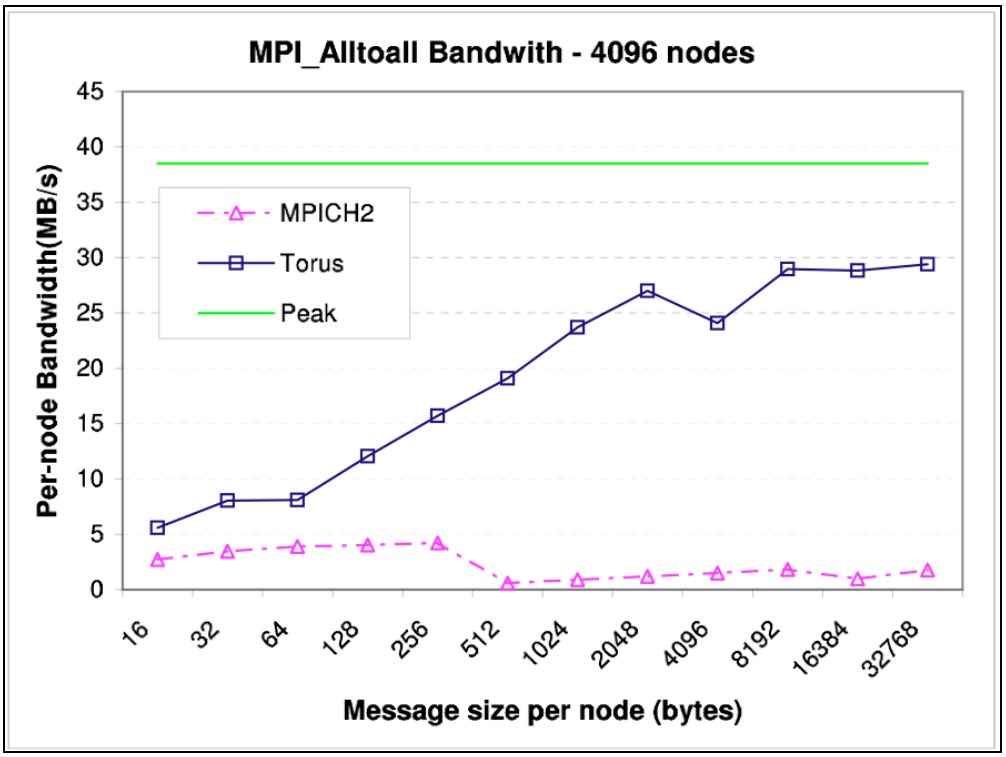

Figure 9. MPI_Alltoall performance on 4096 nodes when sending various sized messages.

10. Summary

The optimized MPI collective routines developed by the IBM BlueGene/L engineers outperform the default MPICH2 implementations in nearly all cases. They achieve this by exploiting knowledge about the torus and tree network topologies, and utilizing special features of these networks such as the deposit bit feature and a built in arithmetic unit.

11. References

[1] The Top 500 Supercomputer Sites. http://www.top500.org. Accessed April 2, 2005.

[2] G. Almási, C. Archer, J.G. Castaños, M. Gupta, X. Martorell, J. E. Moreira, W. Gropp, S. Rus, B. Toonen. "MPI on BlueGene/L: Designing an Efficient General Purpose Messaging Solution for a Large Cellular System", Lecture Notes in Computer Science, vol. 2840, p. 352-61, September 2003.

[3] G. Almási, C. Archer, J. G. Castaños, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Rattermann, N. Smeds, B. Steimacher-burow, W. Gropp, and B. Toonen. Implementing MPI on the BlueGene/L supercomputer. In *Proceedings of Euro-Par 2004 Conference,* Lecture Notes in Computer Science, Pisa, Italy, August 2004.

[4] G. Almási. "BlueGene/L MPI From a User's Point of View." BlueGene/L Systems Software Conference, Salt Lake City, Utah, February 2005.
http://www-unix.mcs.anl.gov/~beckman/bluegene/SSW-Utah-2005/BGL-SSW09-MPIusers.pdf

[5] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *Proceedings of the 11th EuroPVM/MPI conference*. September 2003.

[6] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burrow, T. Takken, and P. Vranas. Design and Analysis of the BlueGene/L Torus Interconnection Network. IBM Research Report RC 23025, Yorktown Heights, New York.

[7] G. Almási, C. Archer, J. Gunnels, P. Heidelberger, X. Martorell, and J. E. Moreira. Architecture and Performance of the BlueGene/L Message Layer. In *Proceedings of the 11th EuroPVM/MPI conference*, Lecture Notes in Computer Science. September 2004.

[8] G. Almási, C. Archer, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, B. Steinmacher-Burow, Y. Zheng. Optimization of MPI Collective Communication on BlueGene/L Systems. IBM Research Report RC 23554, March 4, 2005.

[9] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of Collective Communication Operations in MPICH. In *International Journal of High Performance Computing Applications*, 2005.

[10] N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In *SC2002 – High Performance Networking and Computing*, Baltimore, MD, November 2002.