Optimizing MPI Collectives using Efficient Intra-node Communication Techniques over the Blue Gene/P Supercomputer

Amith R Mamidala[1] Daniel Faraj[2] Sameer Kumar[1] Douglas Miller[2]
Michael Blocksome[2] Thomas Gooding[2] Philip Heidelberger[1] Gabor Dozsa[3]
[1] IBM T.J. Watson Research Center, Yorktown Heights, NY, 10598
[2] IBM Systems and Technology Group, Rochester, MN, 55901
[3] Barcelona Supercomputing Center, Spain
{amithr, faraja, sameerk, dougmill, blocksom, tgooding, philiph, gdozsa}@us.ibm.com

***Abstract*—The Blue Gene/P (BG/P) supercomputer consists of thousands of compute nodes interconnected by multiple networks. Out of these, a 3D torus equipped with direct memory access (DMA) engine is the primary network. BG/P also features a collective network which supports hardware accelerated collective operations such as broadcast and allreduce. One of the operating modes on BG/P is the virtual node mode where the four cores can be active MPI tasks, performing inter-node and intra-node communication.**

This paper proposes software techniques to enhance MPI Collective communication primitives, MPI_Bcast and MPI_Allreduce in virtual node mode by using cache coherent memory subsystem as the communication method within the node. The paper describes techniques leveraging atomic operations to design concurrent data structures such as broadcast-FIFOs to enable efficient collectives. Such mechanisms are important as we expect the core counts to rise in the future and having such data structures makes programming easier and efficient. We also demonstrate the utility of shared address space techniques for MPI collectives, wherein a process can access the peer's memory by specialized system calls. Apart from cutting down the copy costs, such techniques allow for seamless integration of network protocols with intra-node communication methods. We propose intra-node extensions to multi-color network algorithms for collectives using light weight synchronizing structures and atomic operations. Further, we demonstrate that shared address techniques allow for good load balancing and are critical for efficiently using the hardware collective network on BG/P. When compared to current approaches on the 3D torus, our optimizations provide performance up to almost 3 folds for MPI_Bcast and a 33% performance gain for MPI_Allreduce (in virtual node mode). We also see improvements up to 44% for MPI_Bcast using the collective tree network.

I. Introduction

The Blue Gene/P (BG/P) [1] supercomputer is designed to scale to at least 3.56 petaflops of peak performance and is coupled with superior efficiency in the areas of power, cooling and space consumption. BG/P consists of thousands of compute nodes, each composed of four processing cores and arranged as a SMP with hardware managed cache coherency. The compute nodes are interconnected with three different types of networks: (1) a DMA-equipped 3D torus being the principal network for data communication; (2) a collective network supporting hardware accelerated collective operations; and (3) a global interrupt network for hardware accelerated synchronization. Unlike the 3D torus, there is no DMA engine on the collective network.

The primary communication mode on BG/P is by exchanging messages via the standard Message Passing Interface (MPI). One important and widely used mode of executing parallel applications on BG/P is the virtual node mode where all processing cores perform message passing. In this context, not only it is important to consider the inter-node communication but we also need to optimize intra-node communication performance together with the network communication. Moreover, such optimizations are also relevant and crucial in the context of a hybrid model where MPI is used for coarse grain parallelism and OpenMP threads are used for finer grain parallelism of computation. Depending on the characteristics of the application, it may be beneficial to run multiple MPI processes on a node with each process having a certain number of OpenMP threads [2], [3].

In this paper, we focus on optimizing two widely used MPI collective operations namely, *broadcast* and *allreduce* by allowing sharing of data via the cache coherent memory subsystem [1]. Shared memory based methods for these collectives have been extensively studied by many researchers [4], [5], [6], [7], [8], [9]. In these approaches, either a flat or a circular buffer is deployed for staging the data. Data has to be either copied in or out of these buffers during the course of the operation. Synchronization is handled explicitly by setting or reading signaling flags from a shared area. These flags indicate whether a particular data item is either read or written into the data buffers. However, with the rising number of cores per node, effective synchronization techniques coupled with shared data structures are needed to ensure safety and performance at the same time. This is because several MPI processes or threads can potentially access these data buffers. Hence, providing convenient concurrent data structures is extremely important for programmability and performance. Many researchers have explored leveraging atomic operations to design low-overhead, lock-free queues [10]. However, these have been

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.220

explored in the context of point-to-point operations. In this paper, we explore and demonstrate the efficiency of a concurrent broadcast-FIFO for MPI_Bcast using atomic operations by enabling safe enqueue and dequeue of data messages. The FIFO, which is placed in shared memory, can be designed on any platform supporting the basic fetch and increment atomic operation. However, mechanisms employing shared memory involve extra copying of the data from/into the staging buffers.

Recently, several researchers have demonstrated the utility of accessing the peer's memory directly for improving intra-node MPI point-to-point and collective performance [11], [12], [13], [10], [14], [15]. We refer to this approach as the *shared address space*. Processes share their address space directly with other processes via specialized operating system interfaces. The above mentioned studies concentrate mainly on boosting performance within the node. They do not take into account the network features and their interaction with local operations on the node. BG/P provides advanced hardware accelerated collective primitives, both within the torus and the collective tree network. Using the line broadcast feature (deposit bit) of the network, the torus DMA can directly broadcast the data to the application buffers. Further, the collective tree is capable of delivering very low latency and good bandwidth for small to medium messages [16].

Efficiently leveraging these network capabilities in the virtual node mode requires seamless integration of the network protocols with intra-node communication methods. We address this problem and demonstrate various techniques for utilizing shared address space approach to enable high performance collectives. Using message counters and atomic operations, we propose intra-node extensions to the multi-color spanning tree algorithms [16] to obtain the best performance in virtual node mode. All the approaches are generic and can be readily used in RDMA capable networks such as Infiniband. Moreover, we demonstrate that such techniques are critical to obtain good performance on BG/P's collective network. This is due to the fact that there is no DMA support in the collective network and the processing cores need to copy the data in and out of the hardware buffers. The shared address techniques allow for good load balancing across all the cores and enable efficient use of the collective network.

The design of the schemes is integrated into the DCMF messaging stack [17] and CCMI collective framework [18], which are glued to MPICH [19]. When evaluated with micro-benchmarks, the schemes and optimizations in MPI_Bcast and MPI_Allreduce demonstrate significant performance improvement. Specifically, when compared to current approaches on the 3D torus, our optimizations provide performance up to almost 3 folds for MPI_Bcast and a 33% performance gain for MPI_Allreduce (in virtual node mode). We also see improvements up to 44% for MPI_Bcast using the collective tree network.

The paper is organized as follows. We first describe the related work followed by a detailed background of BG/P. We then describe the design details of the collectives MPI_Bcast, MPI_Allreduce over the BG/P 3D torus and collective network in virtual node mode. Finally, we evaluate the techniques proposed in the paper followed by the conclusion and future work.

II. Related Work

Several researchers have studied the benefits of using atomic operations for designing lock-free queues. Buntinas et al. [10] have demonstrated the effectiveness of compare-and-swap atomic operation to link the different data cells to implement a queue abstraction. Though these data structures are efficient to design MPI point-to-point operations, they cannot be used directly for collective operations such as MPI_Bcast. The basic approach used in our techniques is to use the fetch-and-increment operation to design a lock-free broadcast-FIFO, which greatly simplifies the handling of the different queue elements. Also, researchers have shown the effectiveness and applicability of using shared address space techniques or similar mechanisms within a node. Jin et al. demonstrated the kernel aided one copy schemes using LiMIC [13], a kernel module supporting MPI intra-node communication on a Linux machine, integrated into open source MVAPICH [20], [21], [22]. Brightwell et al. have shown the performance gains on the Cray XT using the SMARTMAP [11], [12], [23] strategy developed in the Catamount lightweight kernel. The performance results were shown using open source OpenMPI [24] over Cray. These schemes were also illustrated by using Kaput [25], another kernel module in [14]. However, all these studies concentrated only on the intra-node communication.

In this paper, we explore the advantages of shared address space scheme in designing collectives running over massively parallel supercomputers and involving both inter-node (network) and intra-node communication. In particular, the shared address based methods:

- Avoid extraneous copy costs thereby pushing the performance envelope of the collective algorithms.
- Allow lightweight synchronization structures such as counters to effectively pipeline across the different stages of the collectives: between network and shared memory and across different stages in the shared memory.
- Avoid explicit global flow control across network and intra-node interfaces. Since the destination and source buffers are the application buffers, data is channeled directly in and out of these buffers. Avoiding staging buffers automatically solves the issue of explicit flow control mechanisms. However, care must be taken to pin the buffers in the memory during the operation.
- Since the data synchronization is minimal, the multi-color algorithms can be extended efficiently within the

node. For example, the protocol processing of each color can be done independently by delegating the tasks associated with a given color to a particular node.

However, it must be noted that leveraging the shared address capability is dependent on the interfaces exposed by the operating system. On systems that do not support this functionality, collective algorithms can leverage atomic operations on shared memory based data structures to design efficient MPI primitives.

In this effort, we propose different algorithms and communication mechanisms that leverage the benefits of shared address space and atomic operations.

III. BG/P Overview

BG/P comprises of three different interconnection networks: 3D torus, collective network, and global interrupt network. In this paper, we focus on the 3D torus and collective networks.

A. BG/P interconnection networks

3D Torus: This is the primary network used in BG/P. It is dead-lock free and supports reliable packet delivery. Each node in the torus connects to six neighbors with links of 425MB/s raw throughput. The network provides for hardware accelerated broadcast wherein a deposit-bit can be set in the packet header allowing torus packets to be copied at intermediate nodes along the way to the destination (on torus line). This feature is extensively used in the collective algorithms on BG/P.

Collective Network: The collective network is of a tree topology. It supports reliable data movement at a raw throughput of 850MB/s. The hardware is capable of routing packets upward to the root or downward to the leaves, and it has an integer arithmetic logic unit (ALU). This makes it very efficient for broadcast and reduction operations. Note that there is no DMA engine on this network. Packet injection and reception on the collective network is handled by a processor core [17].

DMA Engine: BG/P improves upon BG/L hardware by adding a DMA engine that is responsible for injecting and receiving packets on the torus network and for local intra-node memory copies. The high performance DMA engine can also keep all six links busy, resulting in better performance of torus point-to-point and collective communication [16], [18], [17].

Direct Put/Get: The DMA engine allows the capabilities of direct-put and direct-get operations to move data to and from a destination buffer. These are similar to the RDMA Write/Read operations of other interconnects such as InfiniBand. In this mechanism, the host posts a descriptor to the DMA with the description of the source and destination buffers. Counters are also allocated to track the progress of the operation which are regularly polled by the processing cores. For every chunk of data read or written, the DMA would appropriately decrement the counter by the number of bytes transferred in the chunk. Together with the torus line broadcast capability, direct-put allows for zero-copy collectives wherein the processor is not involved in any memory copy operations.

B. Shared address technique using Compute Node Kernel (CNK)

CNK is a light weight operating system using a small amount of memory to run and leaving the rest to the application. A very useful feature in CNK is the process window support. By using specialized system calls, the host kernel on BG/P allows for a process to expose its memory to another process. Using this mechanism a process can directly read/write the data from/to the source buffers of another process during the message transfer operations. CNK allows this mapping by using Translation Look-aside Buffer(TLB) slots. By default, the number of TLB slots is set to three, one for each peer process on the node.

IV. Communication Mechanisms

In this section, we first describe the details of data synchronization and transfer mechanisms. These mechanisms are used in designing the various collective algorithms described in the remaining sections of the paper.

We first explain the mechanism of the broadcast-FIFO. We do that by describing the operation of the Point-to-Point FIFO using atomic fetch-and-increment operation which forms the basis of the design of the broadcast-FIFO.

A. Point-to-Point FIFO

By a point-to-point FIFO, we mean a process enqueues a data item into a shared FIFO and only one process dequeues that particular data item. The basic idea in the design of this FIFO is for the memory to be structured in the form of a shared FIFO where a first arriving process reserves a slot in the FIFO followed by the next process and so on. The required attributes of this FIFO are the following: a) Each process enqueues into a unique slot reserved by it. No two processes obtain the same slot in the FIFO. b) Messages are drained in the same order as they were enqueued in. The order of enqueueing is determined by the order of the reservations of the slots.

As shown in Figure 1, enqueueing a data element is accomplished by atomically incrementing the Tail and reserving a unique slot. However, the final location in the FIFO is determined by doing a (mySlot%fifoSize). The atomicity of the counter ensures that no two processes write to the same location in the FIFO. Also, before enqueueing the FIFO must be checked for free slots. The dequeue operation is handled in a similar fashion by incrementing the value of the Head. The particular item is accessed by doing a (Head%fifoSize) for obtaining the index into the FIFO once it is determined that valid data exists at the location pointed to by the Head.

B. Broadcast-FIFO

This FIFO is similar to the point-to-point FIFO for enqueueing the message. As shown in Figure 1, a given process increments the Tail atomically, reserving a unique slot in the FIFO. It proceeds to write data into the FIFO given there is space. The amount of space is determined by checking if (mySlot-Head)< fifoSize. If the condition is true, the data element is enqueued. Also, in addition, an atomic counter variable for this index is set to equal to (n-1) which is the count of all the processes that would dequeue this data element. The enqueue operation is complete when the process finishes the write completion step. The FIFO differs in the way the message is dequeued. Except for the process inserting a message into the FIFO, all the others need to read the message in order for it to be dequeued from the FIFO. The current index of the head is obtained by doing (Head%fifoSize) and after the process reads the message it also decrements the atomic counter, which was initialized to (n-1). When this value reaches zero, the last arriving process completes the dequeue operation and the message is effectively removed from the FIFO by atomically incrementing the Head.

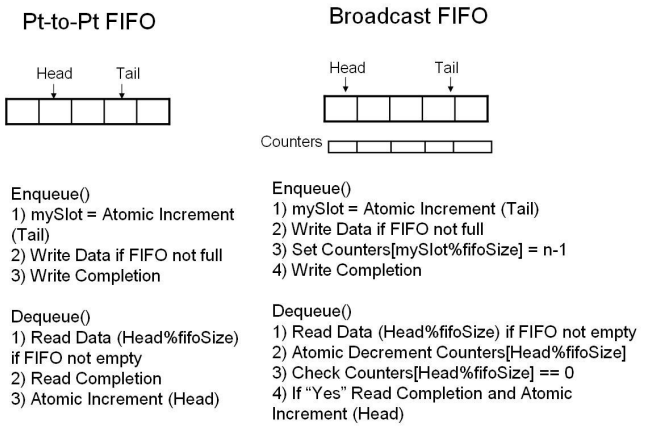

Figure 1. FIFO Mechanisms

C. Message counters with direct copy

Using message counters is a convenient and effective method of tracking the progress of data transfer operations. As explained previously, BG/P extensively uses the counter mechanism to monitor the status of different network operations. We have explored this technique at the software level for doing a direct copy of data in the broadcast operation. The central idea adopted in our approach is to dedicate a counter for a given broadcast and whenever the data arrives in the buffer, it is incremented by the total number of bytes received in the buffer. The detailed usage of these message counters is presented in the next section of the paper.

V. Design of MPI Collectives

In this section, we present the detailed designs of collectives in the context of the shared memory and share address space techniques. The algorithms use the communication techniques explained in the previous section of the paper. Also, note that depending on the message size, either the torus or the collective network based algorithms perform optimally. The torus network is superior for large message collectives where the six torus links together provide more bandwidth than the collective network. However, the collective network is optimal for short to medium messages.

A. Integrated Broadcast over Torus

We first describe the current algorithm that is used in performing large message broadcast over the torus. We then discuss the integration of the intra-node communication mechanisms to provide an efficient broadcast operation.

1) Current Approach: BG/P messaging stack uses *multi-color spanning tree* algorithms for broadcast on the torus network. These algorithms take advantage of the three or six edge-disjoint routes from the root of the collective to all other nodes in 3D meshes and tori, respectively. Each link (connected to a node) can be thought of as a separate *connection* that the node schedules data movement on. Figure 2 illustrates a multi-color rectangle broadcast on a 2D mesh. The root is at the bottom left corner. On a 2D mesh, all root nodes have two edge disjoint spanning trees to all other nodes. In phase 1 of the X-color broadcast, the root sends data first along the X dimension using the deposit bit feature. The X-neighbors of the root then forward this data along the Y dimension in phase 2. On BG/P 3D torus, there are six such spanning trees which correspond to a peak throughput of 2250 MB/s which is close to peak performance. For virtual node mode scenario, an extra fourth dimension is added to the multi-color spanning tree algorithm. This dimension corresponds to the data movement within the node. Also, note that DMA is involved in moving the data across the different phases. Though the DMA is capable of keeping all the six links busy of a 3D torus node, it is not enough to concurrently transfer the data within the node and outside the node (on the network).

2) New Designs: The basic idea behind the designs explained next is to effectively extend the concept of a connection. We provide two techniques for doing this. The first technique streamlines pipelining but does not avoid extra copying. The second technique allows for both. The two techniques differ in how the data gets moved locally once it is received from the network.

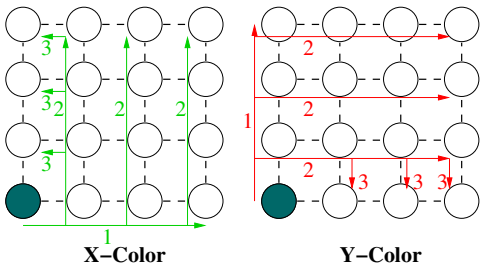

Figure 2. Multi-color rectangle algorithm

Shared Memory Broadcast using Broadcast-FIFO: In this design, the master process forwards the data to its peers using the broadcast-FIFO scheme as follows: once a chunk of data is received from the torus network into the application buffer, the master process enqueues the data element into the broadcast-FIFO. The data is packetized if it is greater than the FIFO slot size. Apart from the actual data, metadata information associated with the data is also copied into the same FIFO slot. The metadata includes the number of data bytes copied into the slot and the connection id of the global broadcast flow. In this fashion broadcast streams from multiple connections can be multiplexed into the same FIFO.

Shared Address Broadcast using Message Counters: The basic idea behind the technique is to streamline pipelined incoming data chunks with ongoing local data transfer from a single process data buffer. We designate the process receiving from the network as the master process. The master after receiving the network data notifies other processes about the arrival of data. The arrived data is copied out directly from the application buffer of the master process using shared address mapping techniques. As shown in Figure 3, a shared counter that is visible to all processes is used for data synchronization. This approach seamlessly integrates with the collective algorithm over the torus network and cuts down the extraneous copy costs. Effectively, the counters mirror the contents of the hardware registers of the DMA as data is written to the application buffers. Note that the buffers shown in the figure are partitioned four ways for data streams flowing from X+, X-, Y+ and Y- directions respectively.

The counter object consists primarily of two fields: (a) base address of the data buffer and (b) total bytes written into the buffer. The master process initializes the counter with the base address of its buffer and also sets the total bytes written to zero. Once the master is notified by the network about the reception of the next chunk of bytes in the data stream, it increments the total bytes by the same amount. The other processes poll the counter value to determine if new data has arrived in the master's buffer. There is also an atomic completion counter which is initialized to zero by the master. All the processes increment this counter after they finished copying the data from the master. Once this counter equals to n-1 where n is the total number of processes, the master can go ahead and start using his buffer. This method is more effective and convenient than the broadcast-FIFO. However, message counters are applicable only to contiguous data flows.

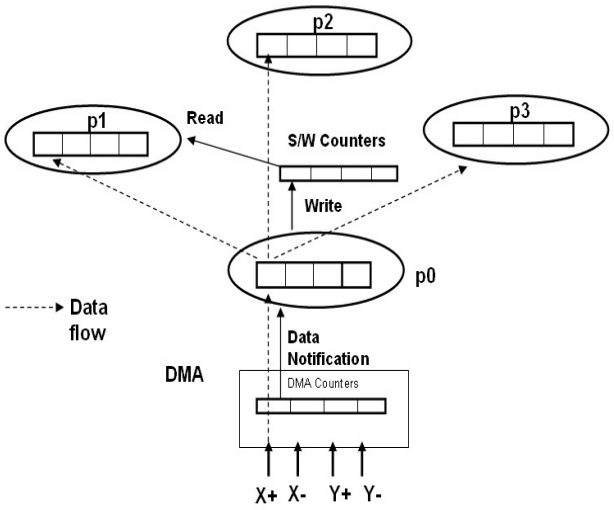

Figure 3. Broadcast using Message Counters

B. Integrated Broadcast over Collective Network

In this subsection, we deal with small and medium message broadcast that uses the collective network. We first explain the current algorithms followed by the new designs proposed in the paper.

1) Current Approach: The current algorithms use the fast hardware allreduce feature (math units) of the collective network. The root node injects data while other nodes inject zeros in a global OR operation. In SMP mode, two cores within a node are required to fully saturate the collective network throughput. Hence, two threads (the main application MPI thread and a helper communication thread) inject and receive the broadcast packets on the collective network. In the virtual node mode, the DMA moves the data among the cores of each node, either via a memory FIFO or direct-put.

2) New Designs: As described above, the tree hardware on BG/P enables a very efficient broadcast mechanism. However, efficiently leveraging this tree requires attention to load balancing across the different tasks. As explained next, shared address techniques are critical to accomplish this load balancing.

Shared Memory broadcast over Collective network:

In this simple and basic design the data from the tree is transferred into a buffer shared across all the nodes. The same core accessing the collective network does both the injection and reception of the data. The received data is placed in a shared memory segment, which in turn is copied over by the other processes on the node. This optimization works for short messages where the copy cost is not a dominating factor in the performance of the collective operation.

Shared Address broadcast over Collective network: to effectively utilize the tree bandwidth when a large message is sent, an injection process injects data into the collective network and a separate reception process copies the network output into the application buffer. However, distributing this data across all the processes in a node poses a problem. Directly using the shared memory techniques creates a scenario where either the injection or the reception process or both are loaded more than the other two processes. Assume that the reception process receives data into a shared memory segment. This data can be copied over by the two idle cores. Yet, both the injection and the reception processes have to simultaneously copy the data into their own application buffers as well. This slows down the injection and reception rate significantly, which degrades the performance. Similar scenarios occur where the reception process receives data directly into its own buffer. Since there is excess of memory bandwidth relative to the tree, the two idle cores can be delegated tasks in the collective operation. We demonstrate the utility of shared address mechanism to solve this problem.

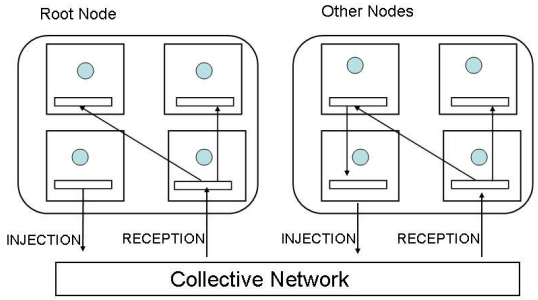

Figure 4. Broadcast using Collective Network

As shown in Figure 4, we delegate different tasks to separate cores in the operation. Assume that the broadcast operation is initiated by the global root whose local rank is 0. We designate all processes with local rank zero from all nodes as the injection processes and the processes with local rank 1 would be the reception processes. All local rank 1 processes receive the data directly into their final buffers. Once a chunk of data is copied into its application buffer, it notifies the other two processes with local ranks 2 and 3. It uses a software shared counter mechanism described earlier. These two processes copy the data directly from the application buffer of the process with local rank 1. Further, the process with local rank 2 makes an additional shared address copy into the application buffer of the injection process. The extra copy is not a problem as the memory bandwidth is at least twice that of the collective network. As we will see later, this approach delivers the best performance.

C. Integrated Allreduce over Torus

We here describe the intra-node multi-color extensions for MPI_Allreduce.

1) Current Approach: The basic idea in this algorithm is to pipeline the reduction and broadcast phases of the allreduce. A ring algorithm is used in the reduction followed by the broadcast of the reduced data from the assigned root process. Similar to the broadcast algorithm, a multicolor scheme is used to select three edge-disjoint routes in the 3D torus both for reduction and broadcast. This scheme is not optimal as redundant copies of data are transferred by the DMA for the reduction operation. Also, the DMA cannot keep pace with both the inter-node and intra-node data transfers. As shown shortly, shared address based messaging overcomes this issue by delegating specialized tasks to different cores.

2) New Design: The allreduce operation can be decomposed into the following tasks: (a) network allreduce, (b) local reduce, and (c) local broadcast. The data is first locally reduced followed by a global network reduction. Once the data is reduced globally on the network, it must be broadcasted locally. The central idea of the new approach is to delegate one core to do the network allreduce operation and the remaining three cores to do the local reduce and broadcast operation. Since there are three independent allreduce operations or three colors occurring at the same time, each of the three cores is delegated to handle one color each. The data buffers are uniformly split three ways and each of the cores works on its partition. The exact mechanism is described below.

Assume that the pipeline unit used for reduction and broadcast used be Pwidth bytes. Once the operation starts, each of the cores performs a reduction on the first Pwidth bytes from each of the four processes application buffers. All application buffers are mapped using the system call interfaces, and no extra copy operations are necessary. The network protocol is exactly identical to its SMP counterpart where there is only one process per node. Once the network data arrives in the application receive buffer of the master core, it notifies the three cores. The other three cores start copying the data into their own respective buffers after they

are done with reducing all the buffer partitions assigned to them.

VI. Performance Study

We evaluate the different schemes on two BG/P racks running in virtual node mode making up a total of 8192 processes.

The micro-benchmark shown in Figure 5 illustrates the way we measured the performance for the broadcast and allreduce operations. Note that the time reported by the benchmark is the maximum among all participants, which captures the total time of the operation and not the local cost of the operation. Also, in all algorithms discussed in the paper, the processes synchronize in order to make sure that the application buffers are ready before the collective actually begins.

```
(1) elapsed_time = 0;
(2) for (i = 0; i < ITERS; i++)
(3)     MPI_Barrier(comm);
(4)     start = MPI_Wtime();
(5)     MPI_Bcast(...);
(6)     elapsed_time += (MPI_Wtime() - start);
(7) elapsed_time /= ITERS;
(8) MPI_Reduce(&elapsed_time, &max, 1, ..);
```

Figure 5. Measuring performance of MPI_Bcast

A. Integrated Broadcast over Collective Network

We demonstrate here the benefits of using shared address and shared memory algorithms for MPI_Bcast over the collective network which provides hardware accelerated broadcast support for MPI_COMM_WORLD. As described earlier, the collective network based algorithms are used for latency sensitive short to medium messages. In the first algorithm named 'CollectiveNetwork + Shmem', a shared memory segment is used to transfer messages from the hardware to the four cores on the node. The second algorithm named 'CollectiveNetwork + Shaddr' describes the shared address algorithm described in the section V. The first algorithm is a latency optimization and is used for short messages. As shown in Figure 6, it provides a 5.83us broadcast latency on 8192 processes. It adds an overhead of 0.4us over the hardware network broadcast, referred to as the 'CollectiveNetwork + SMP' for the intra-node protocol processing. The SMP mode gives the basic network latency as only one process is launched per node, and it directly accesses the collective hardware. Also, the 'CollectiveNetwork + shmem' algorithm improves the performance considerably over the 'CollectiveNetwork + DMA', which uses the DMA to move data within the node after it is received from the collective network. The DMA moves the data to the memory FIFO of the core in this algorithm.

The second algorithm, which is based on the shared address space improves the performance for medium messages.

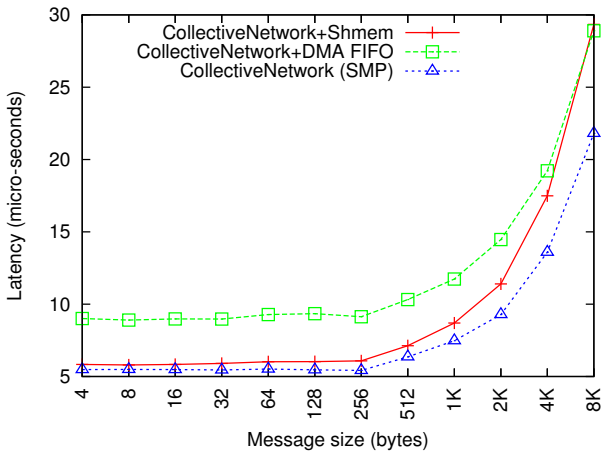

Figure 6. Latency of MPI_Bcast

As shown in Figure 7, the 'CollectiveNetwork+ Shaddr' algorithm outperforms all virtual node mode algorithms. The SMP mode algorithm, 'CollectiveNetwork + SMP' is shown for reference. The other algorithms shown in the figure are the 'CollectiveNetwork + DMA FIFO' and 'CollectiveNetwork + DMA Direct Put', which uses the DMA for moving data within the node. The shared address scheme proposed in the paper improves the bandwidth throughput of medium messages up to 45% for message size of 128K bytes. Figure 8 shows the effect of system call overhead on the performance of the shared address schemes. Note that each mapping of an application buffer involves two system calls: to obtain the physical address from the virtual address and to map the physical address to the virtual address. If these system calls are invoked repeatedly by the application, it contributes to a big source of overhead. In our schemes, we internally cache the buffer information if the same buffer is repeatedly used in the application. Several open source software stacks follow such similar schemes. For example, MVAPICH [22] uses this approach for MPI stacks over Infiniband to avoid the overhead of pinning and unpinning the buffers. Finally, as shown in Figure 9, the algorithm scales well for different process configurations. This is because that the collective network provides very good scalability and performance with increasing number of processes. A detailed study is published in [16].

B. Integrated Broadcast over Torus

In this subsection, we examine the performance of the algorithms designed for medium to large broadcast messages. The first algorithm named 'Torus + Shaddr' is based on the shared address concept described in section V. It uses the light weight message counters to effectively pipeline across network and intra-node communications. The second algorithm named 'Torus + FIFO' uses the concurrent broadcast-FIFO structure described in section IV. These algorithms are

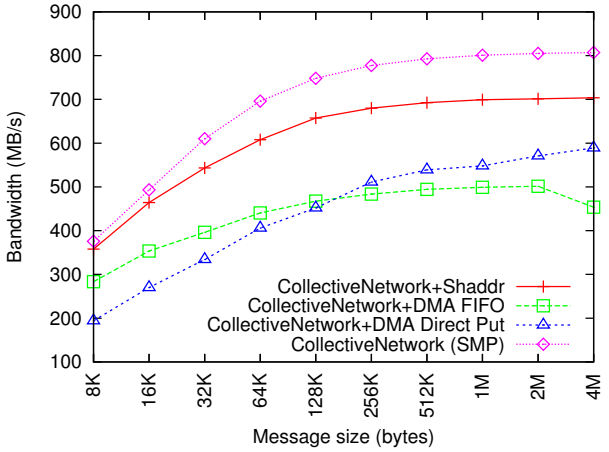

Figure 7. Bandwidth of MPI_Bcast

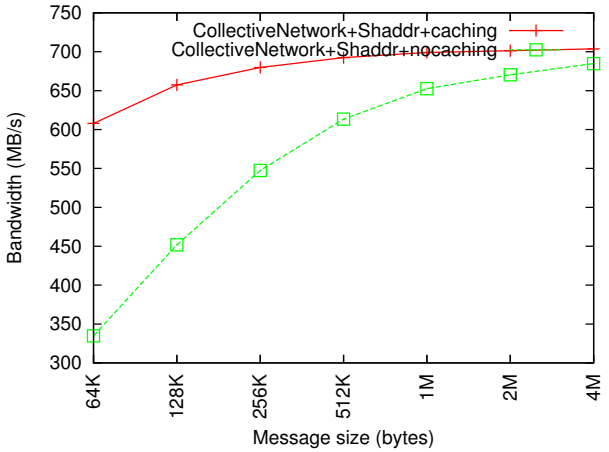

Figure 8. Overhead of the System Call

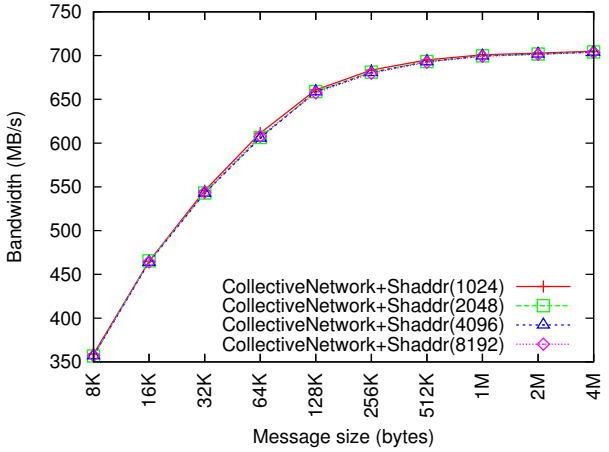

Figure 9. Performance with increasing scale

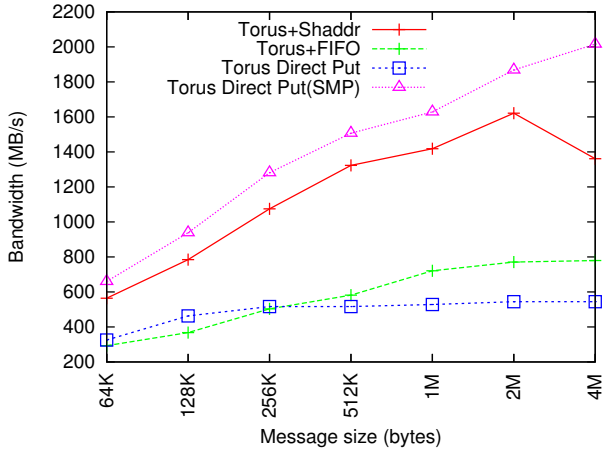

Figure 10. Bandwidth of MPI_Bcast

compared to the current best performing algorithm, 'Torus Direct Put', which involves the DMA to do the message transfer both within the node and as well as outside the node.

As shown in Figure 10, the 'Torus + FIFO' scheme improves the performance of broadcast by a factor of 1.4 for 2MB messages. This is primarily because of concurrent data transfers by the processing cores within a node and the DMA moving the data from the node to the torus network. Note that in this scheme a common FIFO is used to multiplex the data from all the six edge disjoint routes of data flow. As shown in Figure 10, the 'Torus + Shaddr' scheme performs best for large messages. The algorithm is able to achieve a 2.9 speedup at 2M message size. The scheme also gives good improvement for messages at the lower end (64KB messages) and is within 15% of the peak possible for this message in the SMP mode. This is primarily because of the low overhead and light weight message counters which are used for synchronizing the data movement within a node and over the torus. Also, note that the performance drops in the end for the 'Torus + Shaddr' schemes. This is due to the L2 cache size, which is 8MB in size.

C. Integrated Allreduce over Torus

This section demonstrates the performance of the MPI_Allreduce algorithm described in section V. The algorithm uses three cores to exclusively do the reduction and broadcast of data. Each color or connection is assigned to a different core. There is one core dedicated to performing only the network protocol processing. As seen in Table I, we observe performance benefits across the different messages but the algorithm is mostly useful for large messages. As shown in the table, the algorithm provides about 33% improvement for 512K doubles.

Table I
ALLREDUCE THROUGHPUT

Doubles	New (MB/s)	Current (MB/s)
16K	145.35	120.7
32K	191.25	161.5
64K	215.9	196.35
128K	258.4	227.8
256K	289.0	234.6
512K	322.15	241.4

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed software techniques to enhance MPI Collective communication primitives, MPI_Bcast and MPI_Allreduce namely, in BG/P virtual node mode by using cache coherent memory subsystem as the communication method inside the node. The paper proposes techniques leveraging atomic operations to design concurrent data structures such as broadcast-FIFOs to enable efficient collective operations. Such mechanisms are important as we expect the core counts to rise in the future, and having such data structures makes programming easier and efficient. We also demonstrated the utility of shared address space techniques for MPI collectives, wherein a process can access the peer's memory by specialized system calls. Apart from cutting down the copy costs, such techniques allow for seamless integration of network protocols with intra-node communication methods. We propose intra-node extensions to multicolor network algorithms for collectives using light weight synchronizing structures and atomic operation. Further, we demonstrated that shared address techniques allow for good load balancing and are critical to efficiently use the hardware collective network on BG/P as there is no separate DMA engine for moving data into the application buffers. Our performance study reveals the significant benefits of our optimizations, which provide performance up to almost 3 folds for MPI_Bcast and a 33% performance gain for MPI_Allreduce (in virtual node mode) on the torus network as well as 44% speedup for MPI_Bcast using the collective tree network.

In our future work, we intend to extend the mechanism to other collectives such as MPI_Gather and MPI_Allgather, which can also potentially move large volumes of data. Moreover, as shared address mechanisms gain prominence, it becomes important to standardize the interfaces for effectively using these capabilities across different platforms and applications.

VIII. ACKNOWLEDGMENTS

We would like to acknowledge Robert Wisniewski and Craig Stunkel for their useful comments and suggestions. Also, we want to thank the rest of the Blue Gene team. We would like to thank the government for providing the opportunity to work on this project.

REFERENCES

[1] IBM Blue Gene Team, "Overview of the Blue Gene/P project," *IBM J. Res. Dev.*, vol. 52, January (2008). http://www.research.ibm.com/journal/rd/521/team.html.

[2] G. Krawezik and F. Cappello, "Performance Comparison of MPI and three OpenMP Programming Styles on Shared Memory Multiprocessors," in *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2003.

[3] G. Hager, G. Jost, and R. Rabenseifner, "Communication Characteristics and Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-core SMP Nodes," in *Cray User Group Proceedings*, 2009.

[4] R. Graham and G. Shipman, "MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives," in *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2008.

[5] V. Tipparaju, J. Nieplocha, and D. K. Panda, "Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, 2003.

[6] M.-S. Wu, R. Kendall, and K. Wright, "Optimizing collective communications on SMP clusters," in *International Conference on Parallel Processing*, 2005.

[7] A. Mamidala, R. Kumar, D. De, and D. K. Panda, "MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics," in *Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, May 2008.

[8] A. R. Mamidala, A. Vishnu, and D. K. Panda, "Efficient Shared Memory and RDMA based design for MPI Allgather over InfiniBand," in *Proceedings of Euro PVM/MPI*, 2006.

[9] A. Mamidala, L. Chai, H.-W. Jin, and D. K. Panda, "Efficient SMP-Aware MPI-Level Broadcast over InfiniBand's Hardware Multicast," in *Communication Architecture for Clusters (CAC) Workshop*, 2006.

[10] D. Buntinas, G. Mercier, and W. Gropp, "Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem," in *International Symposium on Cluster Computing and the Grid*, May 2006.

[11] R. Brightwell, "A Prototype Implementation of MPI for SMARTMAP," in *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, 2008.

[12] R. Brightwell, T. Hudson, and K. Pedretti, "SMARTMAP: Operating System Support for Efficient Data Sharing Among Processes on a Multi-Core Processor," in *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'08), Austin, TX*, 2008.

[13] H.-W. Jin, S. Sur, L. Chai, and D. Panda, "LiMIC: support for high-performance MPI intra-node communication on Linux cluster," in *Parallel Processing, 2005. ICPP 2005. International Conference on*, 2005.

[14] D. Buntinas, G. Mercier, and W. Gropp, "Data Transfers Between Processes in an SMP System: Performance Study and Application to MPI," in *International Conference on Parallel Processing*, 2006.

[15] D. Buntinas, G. Mercier, and W. Gropp, "Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem," in *Euro PVM/MPI 2006 Conference*, 2006.

[16] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, "MPI Collective Communications on The Blue Gene/P Supercomputer: Algorithms and Optimizations," in *IEEE Hot Interconnects*, 2009.

[17] S. Kumar and et al., "The Deep Computing Messaging Framework: Generalized Scalable Message Passing on the Blue Gene/P Supercomputer.," in *The 22nd ACM International Conference on Supercomputing (ICS)*, 2008.

[18] S. Kumar and et al., "Architecture of the Component Collective Messaging Interface," in *Proceedings of Euro PVM/MPI*, 2008.

[19] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "MPICH: A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, pp. 789–828, September 1996.

[20] L. Chai, P. Lai, H.-W. Jin, and D. K. Panda, "Designing An Efficient Kernel-level and User-level Hybrid Approach for MPI Intra-node Communication on Multi-core Systems," in *Int'l Conference on Parallel Processing (ICPP '08)*, 2008.

[21] L. Chai, A. Hartono, and D. K. Panda, "Designing An Efficient MPI Intra-node Communication Support for Modern Computer Architectures," in *Int'l Conference on Cluster Computing*, 2006.

[22] The Ohio State University, *MVAPICH:MPI over InfiniBand, 10GigE/iWARP and RoCE*.

[23] R. Brightwell, T. Hudson, K. Pedretti, R. Riesen, and K. Underwood, "Implementation and performance of Portals 3.3 on the Cray XT3," in *Proceedings of the 2005 IEEE International Conference on Cluster Computing*, 2005.

[24] *OpenMPI:Open Source High Performance Computing*.

[25] P. Carns, "Kaput, Kernel Module for copying data between processes," 2004.