Efficient High Performance Collective Communication for the Cell Blade *

Qasim Ali
School of Electrical and Computer Engineering, Purdue University
West Lafayette, IN 47907
qali@purdue.edu

Samuel P. Midkiff
School of Electrical and Computer Engineering, Purdue University
West Lafayette, IN 47907
smidkiff@purdue.edu

Vijay S. Pai
School of Electrical and Computer Engineering, Purdue University
West Lafayette, IN 47907
vpai@purdue.edu

ABSTRACT

This paper presents high-performance collective communication algorithms and implementations that exploit the unique architectural features of the Cell heterogeneous multicore processor. This paper specifically describes novel algorithms for the *barrier*, *broadcast*, *reduce*, *all-reduce*, and *all-gather* collective operations, and shows the efficiency of these by comparing them to the previous fastest known implementations of these operations targeting the Cell. The new implementations are faster than the published state-of-the-art, achieving up to 19.21 times the performance (95% reduction in latency) of the previous published collective communication work for the Cell [19, 25]. The results presented show performance both within a chip and across the two Cell chips on a Cell blade [10].

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Parallel programming

General Terms

Algorithms, Performance, Design,

Keywords

Collective communication, algorithms, reductions, Cell processor

1. INTRODUCTION

Accelerator-based computing has proven to be an effective paradigm for achieving high performance, power-efficiency, and space-efficiency, with examples such as the Roadrunner petascale machine that includes Cell processors [1]. A Cell processor contains a single general-purpose 64-bit PowerPC processor (the PPE), which is a dual-issue in-order RISC core, along with eight special-purpose high-performance SIMD processors called synergistic processing elements (SPEs). The first-generation Cell has a peak performance of 204.8 Gflops for single-precision and 14.6 Gflops in double precision mode [12]. The second-generation Cell has 102.4 Gflops peak performance in double precision mode [10].

Cell processor features that can affect the performance and structure of both communication operations and computations are the limited storage that each SPE can directly access (the on-chip *local store*, which has only 256 KB of SRAM per SPE), DMA operations that are the primary mechanism to transfer data, and dedicated hardware communication registers called *mailboxes* and *signal* notification resigters [16]. We explore the use of these features using collective communication operations, since achieving high performance for both kernels and full-scale applications requires efficient communication, and collective communication is a particularly important and time-consuming class of communication [22]. Achieving efficient implementations of collective operations requires attention to algorithmic and hardware issues, and their interplay. This paper shows how designing algorithms and implementations to properly utilize unusual hardware features is extremely important to achieve the highest levels of performance, following in the vein of prior work on platforms like the Cell and BlueGene/L [4, 19, 20].

This paper makes two broad contributions. First, the paper presents detailed descriptions of five new algorithms and their implementations, targeting the common collective communication operations barrier, reduce, broadcast, all-gather and all-reduce. These algorithms are tuned to architectural features and resource limitations of the Cell so as to minimize communication latencies. In describing the implementations in detail, the impact of the on-chip and off-chip interconnects on both concurrency and data traffic in each stage of all of the algorithms is considered. The algorithms are designed keeping in mind the hierarchical network of the Cell Blade, i.e., the fast on-chip network and relatively slower off-chip network. The new strategies include mailbox-based barrier and reductions, optimized binomial broadcast, hybrid segmented binomial broadcast, and hybrid all-reduce and all-gather algorithms to improve performance within the Cell's constraints on concurrency and data traffic. It is our intention that our algorithms (which are currently the fastest we know of) and our use of Cell features will be useful to developers of general applications on the Cell, and to developers of other communication patterns on the Cell. We note that as large numbers of even general-purpose cores are placed on a chip, on-chip bandwidth between cores and bandwidth between chips will increasingly be bottlenecks, leading to trade-offs similar to those we explore on the Cell.

Second, the paper reports performance results for our new algorithms, and compares them to earlier algorithms on the IBM BladeCenters QS20, QS21, and QS22, all of which include two

*This work is supported in part by the National Science Foundation under Grant Nos. CCF-0325603, CNS-0509390, CCF-0532448 and CNS-0751153.

ICS'09, June 8–12, 2009, Yorktown Heights, New York, USA.

Cell processors for a total of 16 SPEs [10]. The results also compare the new schemes against the Cell Messaging Library (CML) and Buffered Mode MPI (BMM), both of which target the Cell [19, 25]. The new algorithms achieve substantial performance improvements against the existing schemes, with performance up to 19.21 times faster than CML (95% less latency) and up to 5.1 times faster than BMM (80.4% less latency) both within a chip and in a blade consisting of two Cell processors. Interestingly, our results also show that mailbox and signal communication registers are not a key to high performance since the Cell supports efficient DMA-based communication between the local store memories of the various SPEs, each inbound mailbox only has a limited number of entries (4), and the 32-bit size of each mailbox entry forces additional messages and ordering constraints to support the 64-bit addresses used in the Cell. The outbound mailboxes and signal registers can hold one entry only. Our new hybrid all-reduce algorithm achieves up to 15% less latency than the current state-of-the-art all-reduce. Our new hybrid all-gather achieves up to 35% less latency than the state-of-the-art all-gather algorithms. The results validate the new communication strategies (including hybrid variants), showing that it is important to consider both concurrency and data traffic when designing on-chip or inter-chip communication algorithms. The implementations described provide the fastest collective communication methods for the Cell at the time of writing this paper.

2. CELL ARCHITECTURE AND COMMUNICATION OVERVIEW

This section describes the Cell and features that are germane to designing efficient collective communication on the Cell. A simplified illustration of the Cell architecture can be found in Figure 1.

2.1 Element Interconnect Bus and Broadband Interface Unit

The PPE and all SPEs communicate through an on-chip high-speed interconnect called the Element Interconnect Bus (EIB). The EIB has a vast amount of data bandwidth (204.8 GB/s) and is the communication path for commands and data between all processor elements on the Cell processor and the on-chip controllers for memory and I/O. It consists of a shared command bus and a point-to-point data interconnect. The command bus distributes commands, sets up end-to-end transactions, and handles coherency. The data interconnect consists of four 16-byte-wide rings, with two used for clockwise data transfers and two for counter-clockwise data transfers. Each ring potentially allows up to three concurrent data transfers, as long as their paths do not overlap. Therefore the EIB can support up to 12 concurrent transfers. To initiate a data transfer, bus elements must request data bus access. The EIB data bus arbiter processes these requests and decides which ring will handle each request.

Each processor element has one on-ramp, and one off-ramp, to the EIB. Processor elements can transmit and receive data simultaneously. Figure 1 shows the unit ID numbers of each element and the order in which the elements are connected to the EIB. The connection order is important to programmers seeking to minimize the latency of transfers on the EIB, as transfers can range from nearest-neighbor (e.g., SPE6 to SPE4) to 6-hop latencies (e.g., SPE1 to SPE6).

The on-chip Cell Broadband Engine interface (BEI) unit provides two interfaces for external communication. One of the two interfaces supports only a non-coherent I/O interface (IOIF) protocol, which is suitable for I/O devices. The other is the Broadband interface (BIF), used for communication between two Cell processors on the same blade. The BIF multiplexes its bandwidth over four rings of the EIB. The bandwidth of the BIF is 80% less than the EIB, so inter-chip communications are much slower than intra-chip communications.

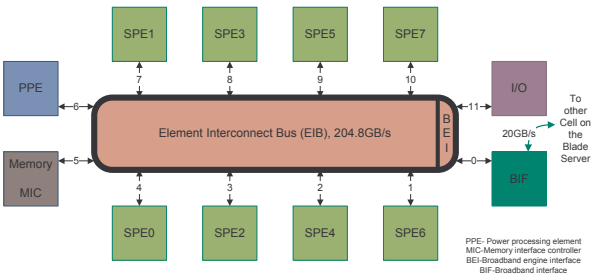


Figure 1: Cell Architecture Overview.

2.2 Synchronization and communication mechanisms

The Cell provides several mechanisms for inter-SPE communication and synchronization. The privileged software on the PPE can map SPE local store addresses and certain MFC resources (e.g., mailboxes) to the main address space, enabling the PPE or other SPEs in the system to access these resources [16]. Such mapping allows each SPE to have knowledge of the other SPEs' local store addresses and can be used for efficient inter-SPE communication or synchronization.

Each SPU also has a set of *mailboxes* that can function as a narrow (32-bit) communication channel to the PPE or another SPE. Each SPU has a four-entry inbound FIFO mailbox. Reading an empty mailbox will block; however, writing into a full mailbox of another SPE will simply result in losing the last entry (the PPE can opt to block on a write to a full mailbox). Each SPU also has *signal* registers, which can also be used for communication.

Simple microbenchmarks show that inter-SPE communication is equally fast for 32-bit quantities transferred using mailboxes, signal registers or memory-mapped local store addresses, with a round-trip time (RTT) of 150 ns. A 64-bit data transfer is equally fast using memory-mapped local store but takes two messages (for a 300 ns RTT) when using mailboxes or signal registers. However, an advantage of using mailboxes and signal registers is that they use dedicated hardware registers, allowing all of the local store to be used for program data rather than synchronization.

3. IMPLEMENTATION OF COLLECTIVE COMMUNICATION ALGORITHMS

Targeting synchronization and collective communication operations for the Cell requires a careful consideration of the available communication mechanisms, the concurrency allowed by any specific algorithm, and its impact on interconnect traffic. This section describes, in detail, five new algorithms and and their implementations to support efficient *barrier*, *reduce*, *all-gather*, *broadcast*, and *all-reduce* operations. We compare these, and other algorithms we developed (but do not describe in detail for space reasons), to a range of existing algorithms in Section 4.

3.1 Setup and Notifications

In all the algorithms, the PPE spawns the threads (using the `pthreads` library) on the SPEs and then determines the physical SPE ID for each running thread. All communication is based on the physical SPE ID rather than the logical thread ID, since communi-

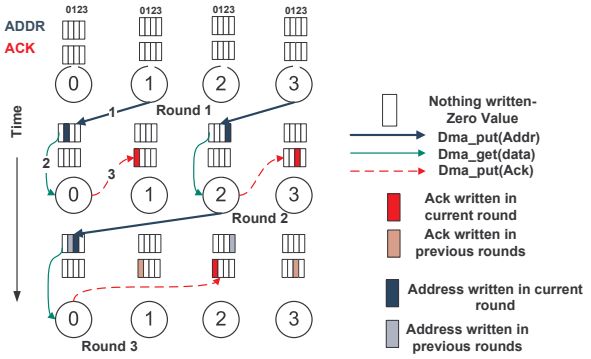


Figure 2: Local store based communication.

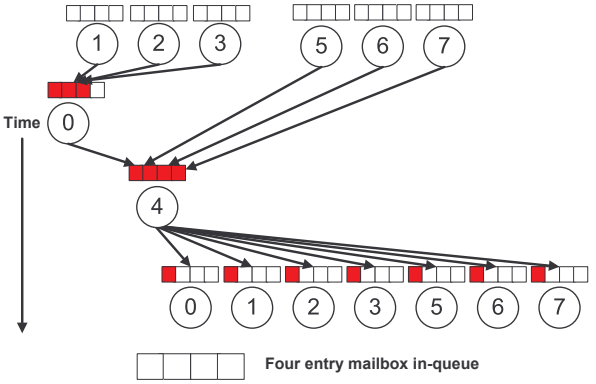


Figure 3: Mailbox based Barrier algorithm for 8 SPEs.

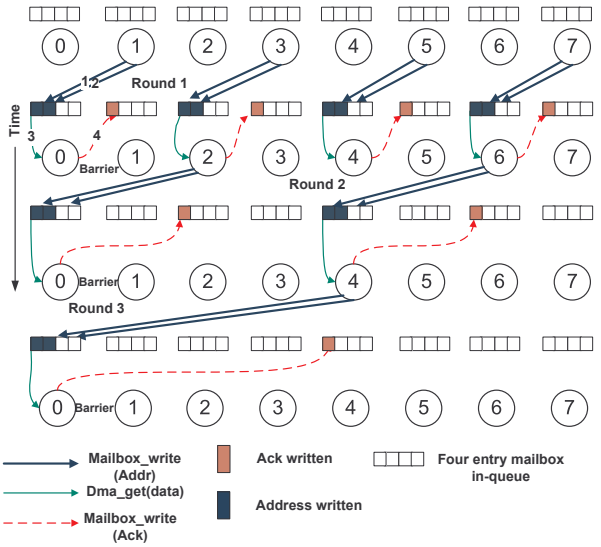


Figure 4: Mailbox-based binomial reduce.

cation latencies depend on physical SPE assignment. This policy ensures that an algorithm achieves consistent results regardless of the resource allocation decisions made by the scheduler. In all the figures, numbers inside a circle represent the physical SPE ID, corresponding to the locations on the chip shown in Figure 1.

Following the technique mentioned in Section 2, privileged PPE software maps the SPE local stores and MFC resources into the address space. The PPE code then sends these addresses to all of the SPEs so that they can communicate via inter-SPE DMAs without going through the PPE or main memory. The PPE is no longer involved once the threads are initiated.

Almost all the algorithms require pairs of SPEs to establish some notification protocol to announce when an SPE is ready to provide data or when an SPE has completed a data transfer. Mailbox messages may be used for this purpose, with the caveats (mentioned in Section 2) that the number of messages is limited and that each message is only 32 bits. Alternatively, SPEs may maintain notification arrays in their local store. Each notification array has x entries, where x is the number of SPEs participating in a particular collective. Each notification array entry uses 16 bytes and is written into by atomic DMAs. These arrays are used for address (ADDR) communication and acknowledgments (ACKs) between the SPEs in a lock-free manner [9]. Figure 2(a) shows the working of these for a simple binomial-tree based reduce. Although notification arrays allow general communication patterns, they do consume a non-negligible amount of local store; each 16 SPE collective typically requires an address array and an acknowledgment array, for a total of 512 bytes. Codes with multiple, possibly overlapping, collective calls must have several such arrays, which can be significant given the capacity limitations of the local store. In contrast, mailbox and signal communication use dedicated hardware registers and consume no local store.

3.2 Limitations of Mailbox and Signal Communication

To explore the impact of mailbox and signal registers on collective communication, we implemented a barrier using signal registers and mailboxes and a reduction using mailboxes.

Barrier implementation. To achieve a barrier, each SPE can send an acknowledgment (ACK) message to one particular SPE, say SPE 0. Once SPE 0 gets the ACKs from all participating SPEs, it sends an ACK to the SPEs indicating that all the SPEs all synchronized. Using mailboxes in this solution, however, will not work with more than four SPEs because each SPE has only four entries in its read inbound mailbox queue. Nevertheless, by performing the barrier in steps, a barrier can be implemented over all SPEs without overflowing any mailboxes. Figure 2(b) shows eight SPEs synchronizing among themselves. The signal registers allow a method to avoid the mailbox overflow problem using their "wired OR"-mode. Thus, we can implement a one-to-all (OTA) barrier using signal registers.

Reduction implementation. We also implemented mailbox-based (using inbound mailbox) reductions in which the ADDR and ACK transfers described in Section 3.1 are done using mailboxes rather than arrays in the local store. As shown in Figure 4, the ACK is a single message, but the 64-bit address takes two mailbox messages; these are ordered using a fence. Beyond this, however, there must also be some mechanism to enforce atomic 64-bit writes into each mailbox. For example, if node n_i writes 32 bits of an address to a mailbox, but node n_j writes into the same mailbox before n_i can write the other half of the address, the mailbox entries for the address being specified by n_i would be separated and the address would be corrupted. Such interleavings can be avoided by adding a barrier between each step of the reduction so that nodes that have already completed a round will not advance to the next round and possibly interfere with nodes that have not yet completed the earlier round. This barrier also eliminates any possibility of mailbox overflow since there are only two messages written to each mailbox

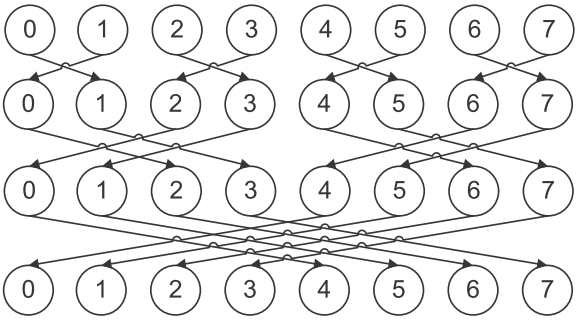


Figure 5: Recursive distance doubling (rdb) algorithm.

between barriers. The cost, however, is increased latency caused by multiple barriers during a reduction.

Both mailbox-based implementations thus see additional constraints caused by mailbox capacity and ordering requirements when compared to communication via local store. Section 4 evaluates the impact of these considerations. Signal registers have limited applicability and also have many of the same contraints as mailboxes.

3.3 All-gather Algorithms

All-gather semantically consists of a gather followed by a broadcast. Implementing it as such would take $2 \times \log_2 p$ communication rounds for a power-of-two number of nodes p. In contrast, the highly concurrent recursive doubling (rdb) all-gather completes in $\log_2 p$ rounds by exploiting full-duplex communication [22]. Figure 5 shows rdb with 8 nodes, with each stage requiring 8 data messages. Note that for non-power-of-two p, rdb requires $\lfloor \log_2 p \rfloor + 2$ rounds to pass information from and send results to the excess nodes.

We have developed a hybrid version of the all-gather collective for the sixteen SPE (two chip) scenario. In this case, rdb requires sixteen data messages in each stage. If the lower-numbered SPEs are on one chip and the higher-numbered ones are on another, the last round will require sixteen data messages to cross the Broadband Interface (BIF), which has far less bandwidth than the on-chip EIB. (If the SPE numbers are spread out between the chips, there will be BIF traffic at every round.) In contrast, using gather plus broadcast only requires two data messages to cross the BIF: one at the end of the gather and one at the beginning of the broadcast. Previous works argue that the number of BIF messages should be minimized [5, 19]. However, doing so may result in a loss of concurrency, resulting in more communication rounds and ultimately poorer latencies.

This paper presents a hybrid algorithm that seeks to find an ideal compromise between the concurrency of rdb and the low traffic of the gather/broadcast combination. For simplicity, consider the case of p nodes where p is a power of 2. This algorithm starts by performing some k rounds of binomial gather, leaving $\frac{p}{2^k}$ evenly-spaced node IDs in the communication structure. At this point, the algorithm switches to rdb all-gather until all of those $\frac{p}{2^k}$ nodes have the results of all-gather. This takes $\log_2 p - k$ rounds and results in $\frac{p}{2^k}$ messages across the BIF if the lower half and upper half of the nodes are on separate chips. Now, each of the $\frac{p}{2^k}$ nodes with the result act as the root of binomial broadcast trees, requiring k more rounds to give the results to all nodes. Thus, the total number of rounds are $\log_2 p + k$ and there are $\frac{p}{2^k}$ BIF-crossing data messages.

Figure 6 shows this algorithm working with 16 SPEs and switching to the rdb algorithm in the fourth communication round ($k = 3$). For simplicity, this figure only shows the data messages; the address exchanges and acknowledgments are handled as described in Section 3.1. This algorithm starts with three rounds of binomial gather and then switches to the rdb algorithm in the fourth round. The number of data messages crossing the BIF is two. At the end of the third round, SPE 0 has the left half of its receive buffer filled with data from SPE0 to SPE7 and the right half is empty. Similarly, SPE 8 has its right half filled up. At the end of fourth round, SPEs 0 and 8 have the results of the all-gather (shown as dark circles in the Figure 6), and each will act as the root of an eight node sub-binomial-tree. An eight node binomial-tree broadcast finishes in three rounds. Hence with a total of seven rounds and two data messages across the BIF, the all-gather is complete with all the nodes having the result of the all-gather.

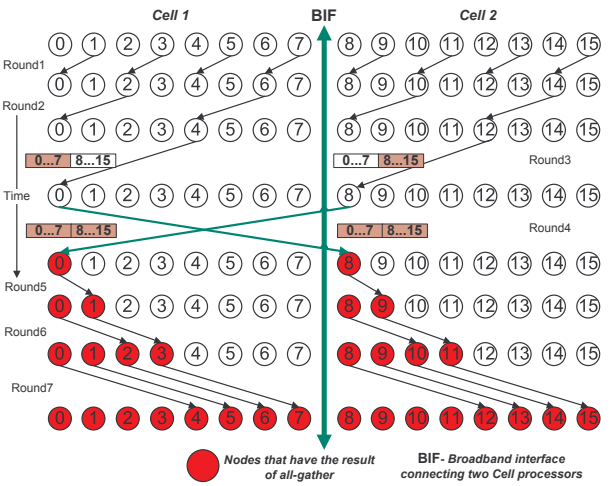


Figure 6: Hybrid 4-2 all-gather (switching round=4,data messages across BIF=2).

This paper evaluates three realizations of this hybrid algorithm, for 16 SPEs and $k = 1, 2, 3$. We call these hybrid-2-8, hybrid-3-4, and hybrid-4-2, respectively. The first number indicates the round in which the algorithm switches from the binomial tree gather to the rdb. The second number indicates the number of BIF-crossing data messages. These three versions take five, six, and seven communication rounds, respectively. Note that hybrid-1-16 corresponds to a full rdb algorithm.

3.4 Broadcast Algorithms

We implemented optimized versions of two-to-all, binomial tree, k-chains (k fanout followed by k chains [21]), segmented binomial tree broadcast, and hybrid segmented binomial for the two-chip (16 SPEs) scenario. Although there is nothing semantically to stop a node from sending data addresses to all of its children without waiting for acknowledgments, this may hurt performance. In particular, it is important that child nodes with more successors complete their DMAs before child nodes with fewer successors or leaf nodes. Otherwise, an entire branch of the tree may have to delay waiting for unrelated nodes to complete their DMAs. For example, if SPEs 1, 2, and 4 in the binomial broadcast shown in Figure 7, all get SPE 0's address one after another, it is possible that SPE 2 could tie up the DMA capacity of SPE 0's local store before SPE 1 gets a chance to perform its DMA. This in turn delays SPEs 3, 5, and 7, all of which depend on SPE 1. To avoid this *priority inversion* problem, broadcast uses address-data-ACK phases at each round as described in Section 3.1. As shown in Figure 7, at the root level, SPE 0 first does a dma_put of the data address into the dark-colored SPE 1 and waits for an ACK from it to indicate

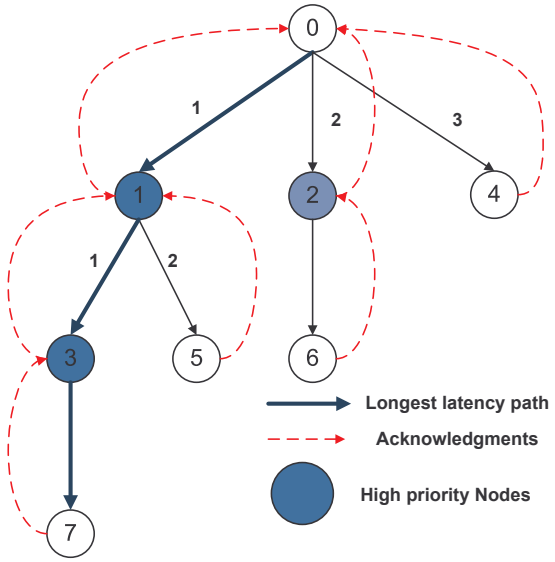


Figure 7: Optimized priority-based binomial-tree broadcast

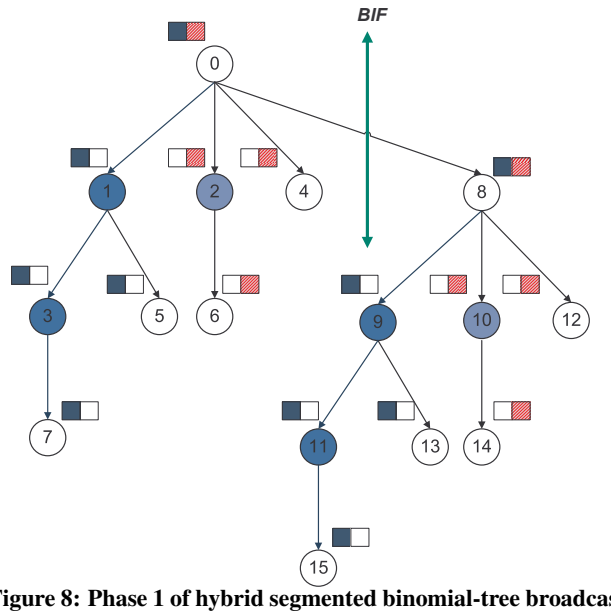


Figure 8: Phase 1 of hybrid segmented binomial-tree broadcast

data transfer completion. After that SPE 0 does another dma_put into the light-colored SPE 2, and so on. (Section 4 also evaluates broadcast without waiting for ACKs at each round.) We also optimized two-to-all, k-chains, and segmented binomial using the same priority optimization.

Segmented binomial tree. For large data sizes, we implemented a segmented binomial tree which consists of two phases. In the first phase, the left half of the tree gets the left half of the data and the right half of the tree gets the right half. In the second phase, the left and right half nodes exchange their data. If the total number of nodes is even, then the left subtree will have one more node than the right subtree. In that case, the root node will provide the last node on the left subtree with the right-half data.

For the 2-chip (16 SPE) scenario, for all the broadcast implementations the first communication round is between the root SPE on one chip and a partner on the other chip; from that point on, each of the two chips separately follows the broadcast tree for its 8 SPEs, except for the segmented binomial, in which there are inter-chip exchanges in the second phase.

Hybrid segmented binomial tree. For 16 SPEs, we implemented a hybrid version of the optimized segmented binomial, in which the root node sends all data to one representative node in the second chip. Then the root node and the representative node on the other chip do the same two-phase process as explained above for the intra-chip (8 SPE) case. In Figure 8, the left half of the data is shown as dark color (blue) and the right half is shown in light color (red). Note that the number of messages crossing the BIF is one in the first phase. In the second phase all the exchanges are intra-chip. If the 16 SPE segmented binomial is implemented without this optimization then the nodes on the left and right half would exchange data in the second phase. All those data exchanges would be inter-chip, resulting in 14 BIF-crossing messages.

3.5 All-reduce Algorithms

This paper considers three new forms of all-reduce. The first is a specialized rdb that includes barriers to avoid overlapping data transfers on the EIB. Then we also considered two novel forms of all-reduce, HybridA and HybridB for 16 SPEs. HybridA is based on binomial reduce, rdb all-reduce using a subset of the nodes, and binomial broadcast. HybridB is based on reduce-scatter (described below) and hybrid all-gather.

A strictly rdb all-reduce completes in $\log_2(p)$ steps since each step has all nodes exchanging data with different partners. However, this causes a great deal of data traffic on the EIB and multiple DMA accesses to the local stores of each SPE. EIB and local-store contention can be controlled by inserting a barrier periodically. The reason for choosing an inter-barrier separation of two rounds is as follows: after the first round there are eight data messages over the EIB (each SPE is doing a data transfer); after the second round there are $16 = 8 + 8$ data messages that have ever been initiated on the EIB. Some of those transfers might not have been completed by the end of the second round, so there is a large probability that there would be more than twelve transactions in flight or fewer than twelve but overlapping transactions. As indicated in Section 2.1, the EIB only supports up to twelve concurrent non-overlapping transfers. So, such a scenario will stress the EIB and serialize the remaining data transfers. Note that ACK and address messages also require time on the EIB; however, these messages are small and finish early. Thus, they are unlikely to cause any serious EIB contention.

We have developed two hybrid versions of the all-reduce collective for the sixteen SPE (two chip) scenario. We call these HybridA and HybridB algorithms, and they are structured similarly to the hybrid all-gather of Section 3.3.

Figure 9 shows HybridA algorithm working with 16 SPEs and switching to the rdb algorithm in the third communication round ($k = 2$). For simplicity, this figure only shows the data messages; Section 3.1 describes address and acknowledgment handling. HybridA starts with two rounds of binomial reduce and then switches to rdb all-reduce in the third round. The number of data messages crossing the BIF is four. At the end of fourth round, SPEs 0, 4, 8 and 12 have the results of the reduction shown as dark circles in the Figure 9, and each of them will act as the root of a four node binomial-tree. A four node binomial-tree broadcast finishes in two rounds. Hence the all-reduce is complete at all nodes with a total of six rounds and four data messages across the BIF.

The HybridB algorithm first uses Rabenseifners reduce-scatter [22], which is an efficient algorithm optimized for large data sizes and is based on recursive vector and distance halving.

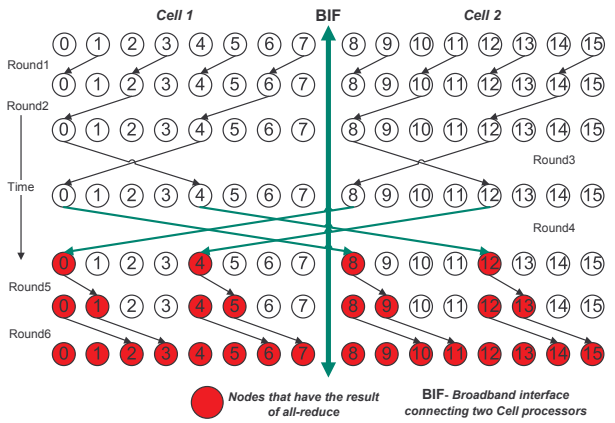


Figure 9: HybridA-3-4 all-reduce (switching round=3,data messages across BIF=4).

Reduce-scatter effectively decreases the aggregate bandwidth requirement by giving each node only a subset of the reduced data. For a power-of-two number of nodes p, at every step k, the node of rank r exchanges data with node of rank r XOR $2^{\log_2(p)-k}$ and the number of exchanged data blocks is halved at every step. Then, HybridB invokes the hybrid all-gather algorithm dicussed in Section 3.3. HybridB-x-y means that it uses hybrid-x-y all-gather. Similar to hybrid all-gather, we have 3 realizations of hybridA and hybridB all-reduce.

4. EXPERIMENTAL EVALUATION

We used IBM's Cell SDK3.0 for coding all the algorithms. All experiments reported here were performed on the IBM BladeCenters QS20 at Georgia Tech (the same platform used for Buffered Mode MPI [25]), as well as QS21 and QS22 blades [10]. The results obtained were similar on all of these servers as our implementations are based on inter-SPE data transfers only, do not use the PPE after initialization, and do not use double-precision floating point. Timings are measured using a fine-grained decrementing register provided by the Cell blade. All latency numbers were averaged over 100,000 iterations.

These results compare the implementations described in Section 3 to the publicly-available Cell Messaging Library (CML). The code for the new implementations and benchmarks can be downloaded from http://www.ece.purdue.edu/∼qali/CellCode. These results also compare the new algorithms to the Buffered-Mode MPI (BMM) numbers reported by Velamati et al. [25]. As their communication code is not available, the comparison charts in this section use the quantitative results from the BMM papers, consisting of only a few numbers for barrier, broadcast, and reduce [17, 18, 25]. All plots are shown on a log-log scale unless otherwise noted.

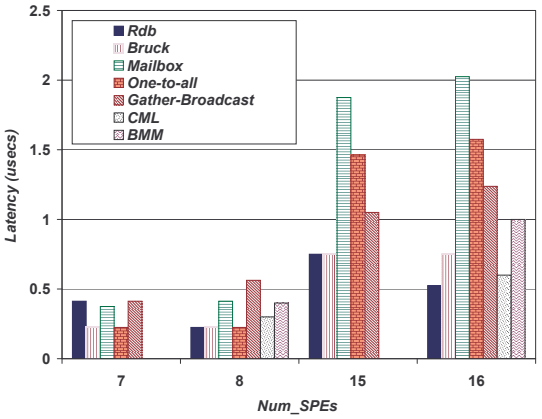
Barrier latency. A centralized atomic increment barrier has a latency of 35 microseconds for 8 SPE's. OTA barrier using OR-mode signotify registers has a latency of 400 ns. For 8 SPEs, the cost of rdb, One-to-all, and Bruck is only 225 ns, which is 25% less latency than CML and 43.75% less than BMM. There is also little room for further improvement, since a (non-tight) lower bound of the barrier cost would be the RTT of 32-bit signals, DMAs, or mailbox messages, which is already 150 ns. The CML barrier implementation uses rdb. In One-to-all, all nodes communicate directly with the root SPE. The OTA signal register implementation is slightly slower than the OTA DMA based implementation because of some extra control and computation code, such as the OR operation. In Bruck, at step k, process r sends a message to rank (r+2^k) and receives message from rank (r-2^k) with wrap around [8, 21]. Bruck always has $\lceil \log_2 p \rceil$ rounds for p nodes. For non-power-of-two number of SPEs less than eight, Bruck and One-to-all are the best, since rdb's cost increases due to an extra round. For 16 SPEs, the rdb barrier is the best because inter-chip messages are exchanged only in the last round of the algorithm. In Bruck, 2^{round_num} inter-chip messages are exchanged at each round, leading to higher per-round latency. For 15 SPEs, both Bruck and rdb have the same latency because the inter-chip message penalties of Bruck offset the extra round seen in rdb. OTA performance is worse for more than 8 SPEs because all of the BIF-crossing messages are coming from a single source. Our rdb barrier implementations for 16 SPEs complete in 525 ns, for 12.5% less latency than CML and 47.5% less than BMM. Note that Figure 10(a) has a linear scaled Y-axis.

All-gather latency. Figure 10(b) shows the latency of various all-gather implementations as a function of data size. For 8 SPEs, all-gather by recursive doubling outperforms gather plus broadcast for all data sizes, as the EIB can handle the data traffic generated by the rdb all-gather. However for 16 SPEs, these two solutions have similar performance for large data sizes (see 32K data point in the Figure 10(b)). Although gather plus broadcast requires twice as many communication rounds, it only requires two messages to cross the inter-chip BIF (one in gather and one in broadcast). Rdb requires 16 messages to cross the BIF in the last round as each node communicates with its peer on the other chip. This cost becomes significant for larger data sizes, motivating the use of customized hybrid all-gather algorithms as discussed in Section 3.3. Figure 11(b) shows the performance of hybrid all-gather variants for data sizes larger than 2 Kbytes; rdb performs best for smaller data sizes, as shown in Figure 11(a). At 4 Kbytes Hybrid-2-8 beats rdb as it cuts EIB traffic in half. After 4 Kbytes Hybrid-3-4 starts performing better than all variants because it reduces the total BIF traffic considerably and has the best level of concurrency. Recall that we consider different variants of hybrid implementations to see the point where the number of data messages crossing the BIF and the concurrency (in terms of number of rounds) combine to form the best result. Hybrid-3-4 achieves the best performance, with up to 35% less latency than rdb-allgather. Note that Figure 11 has a linear scaled Y-axis.

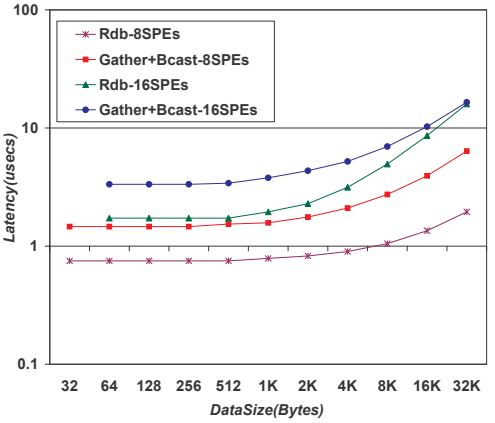
Reduce latency. Figure 12 shows reduce latency as a function of the data size in bytes, for both 8 and 16 SPE systems. All reduction experiments use integer arrays and SIMD-ized vector addition on the data. Both the local-store based and mailbox reduce implementations outperform CML and BMM. The CML implementation uses a binomial tree reduce. Comparisons to BMM are available only for 128 bytes and 1 KB; the new implementations are respectively 5.1 and 3.5 times faster than BMM for 8 SPEs For 16KB and 32KB, the speedup relative to CML reaches a factor of 19. Similar performance gaps are seen with 16 SPEs. The reason for CML's high latency in reductions is that CML has some additional data copies not used by our implementation.

Rabenseifner's approach of using reduce-scatter followed by gather outperforms binomial tree reduce for larger data sizes (greater than 1KB for 8 SPEs and 4KB for 16SPEs). Binomial tree reduce has less concurrency in data traffic and computation with each successive round; in contrast, reduce-scatter exploits link and computational concurrency at each round as the data is recursively halved. The following gather adds rounds and some latency, but is offset by the increased concurrency during the reduce-scatter.

The latencies of the binomial local-store based and mailbox implementations described in Section 3 differ only by a constant amount of time regardless of data size: this is the cost of doing an extra mailbox message for 64-bit addresses and cost of the barriers

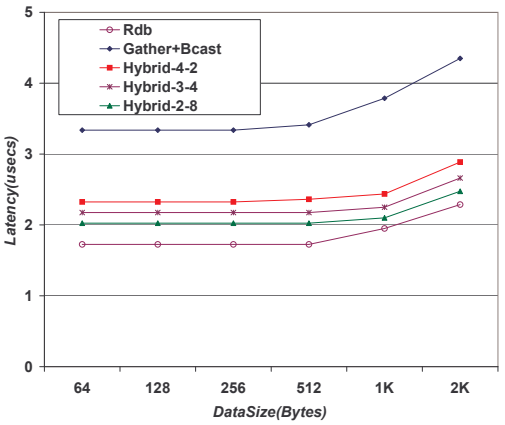


(a) Latency of Barrier.

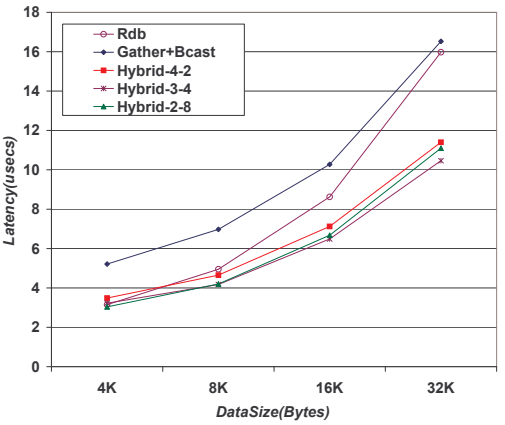


(b) Latency of all-gather.

Figure 10: Performance of Barrier and all-gather.



(a) Latency for small data sizes



(b) Latency for large data sizes

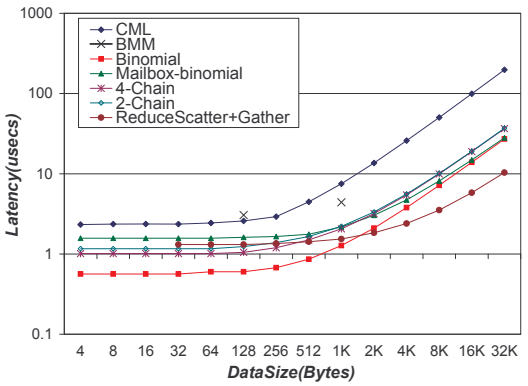
Figure 11: Comparison of hybrid all-gather with simple all-gather algorithms.

after every round. This gap makes up a relatively large fraction of latency for small data sizes, but is nearly inconsequential for larger data sizes. Thus, mailbox-based implementations may be viable for large data sizes if local store capacity is already fully utilized. Otherwise, however, the constraints imposed by mailboxes negate any benefit of having dedicated hardware resources for communication.

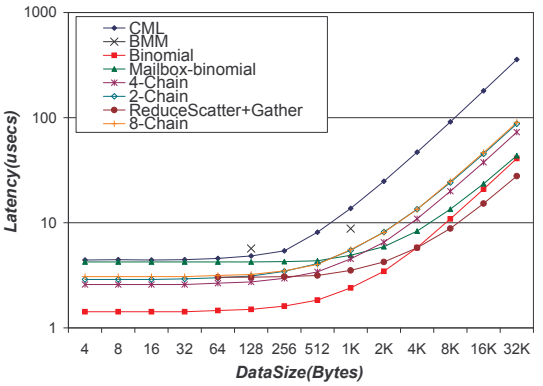
Broadcast latency. Figure 13 shows the latency of the broadcast collective, comparing several versions: CML and BMM represent the prior art, while One-to-all, Two-to-all (root first sends data to another SPE and then those two SPEs send to all the others), Binomial-Tree , k-chains (k fan-out followed by k chains) and segmented binomial and their optimized versions represent the implementations described in Section 3.4. CML implements broadcast using a binomial tree. For all new implementations, the broadcast benchmark uses a barrier after every broadcast call to ensure that the communication of successive broadcasts are not overlapped, leading to overly optimistic performance measurements. The cost of the barrier is included in the broadcast latency results. Barriers were not needed in CML, as the CML implementation itself prevents the overlapping of broadcasts.

For binomial tree, "simple" depicts a broadcast in which nodes do not wait for acknowledgments at the end of each communication round, possibly allowing later communication rounds to complete their data transfers before earlier ones, while "opt" is the optimized version that prevents priority inversion by ordering DMAs. Despite adding ordering requirements and acknowledgments, the *opt* versions outperform *simple* for all broadcast variants (not all shown to avoid graph clutter), achieving up to 42% less latency.

For both 8 and 16 SPEs, our best algorithms outperform CML and BMM. For 8 SPE's, one-to-all performs best for smaller data sizes up to 2 KB, as these traffic sizes have little EIB contention and thus benefit from the increased simplicity. One-to-all underperforms the slightly more concurrent two-to-all for 4 KB data size, but still outperforms binomial tree. Above 2–4 KB, the segmented-binomial tree tree performs best. The first phase of segmented binomial has as many rounds as pure binomial, but each round has less latency because its data size is only half the total. The second phase has a single-round fully concurrent exchange between the two subtrees; for large data sizes, the cost of this extra round is less than the savings achieved by lower latency in the first phase.

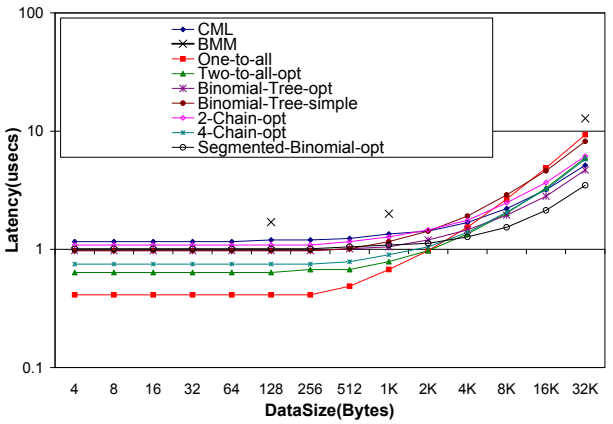


(a) 8 SPE's

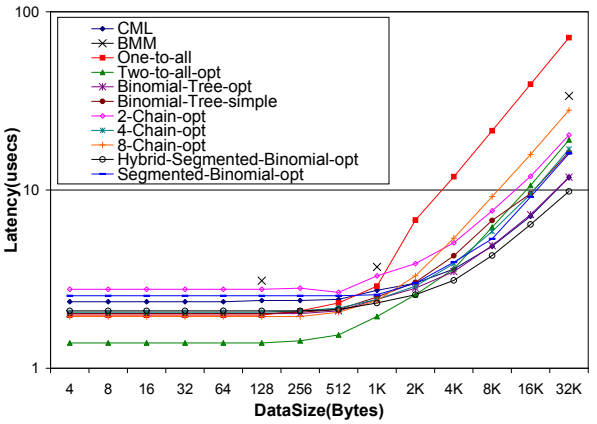


(b) 16 SPE's

Figure 12: Reduce Latency



(a) 8 SPE's



(b) 16 SPE's

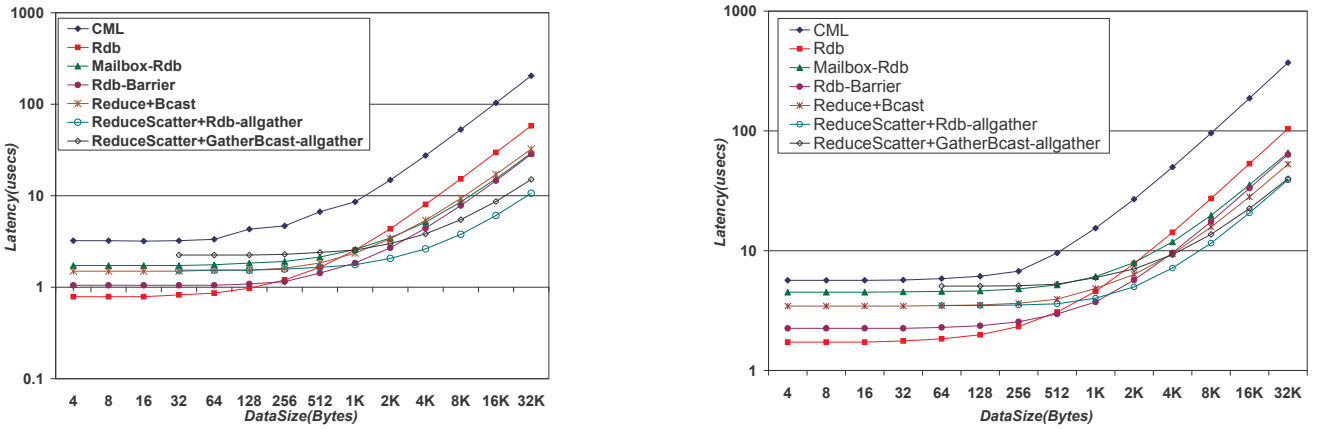
Figure 13: Broadcast Latency

For 16 SPE's, one-to-all is never the best; instead, two-to-all is best for small message sizes. This is because in one-to-all, 15 data messages are crossing the BIF, which has much lower bandwidth than the EIB. Recall that in two-to-all (assuming $spe0$ is the root), $spe0$ first sends the address of its data to $spe8$, which is across the BIF, and waits for it to complete; after that, $spe0$ and $spe8$ do one-to-all broadcasts within their respective processor chips. Above 2KB, the hybrid segmented binomial tree gives the best performance and outperforms all other algorithms. This algorithm has just one BIF-crossing message at the beginning of the first phase of the broadcast. The remainder of the first phase and all of the second phase use only intra-chip data exchanges. In contrast, a non-hybrid 16 SPE segmented binomial would result in an exchange of 14 messages over the BIF, including BIF transfers in both the first and second phases. Its performance is thus worse than even an unsegmented binomial tree.

All-reduce latency. Figure 14 shows all-reduce latency for different implementations: CML, rdb using local-store and mailbox notification alternatives, rdb algorithm using barriers, reduce plus broadcast, and reduce-scatter plus all-gather variants. CML's all-reduce implementation uses binomial reduce followed by binomial broadcast.

As explained in Section 3, the mailbox implementation of all-reduce has a barrier at the end of each round. Figure 14(a) shows that the mailbox implementation has a constant overhead above local store notification (the cost of an extra mailbox message for each address plus 3 barriers for the 8 SPE case), but that this only applies for data sizes up to 256 bytes. Beyond that point, the local-store notification performance degrades, because the EIB is stressed for data sizes larger than 256 bytes. This behavior does not arise in the mailbox implementation because the barrier at the end of each round naturally prevents EIB saturation. As explained in Section 3.5, the EIB can only support 12 concurrent non-overlapping data transfers, so it may be advantageous to include a barrier at the end of every 2 rounds in the local-store version. This implementation is represented in Figure 14 as "Rdb-Barrier".

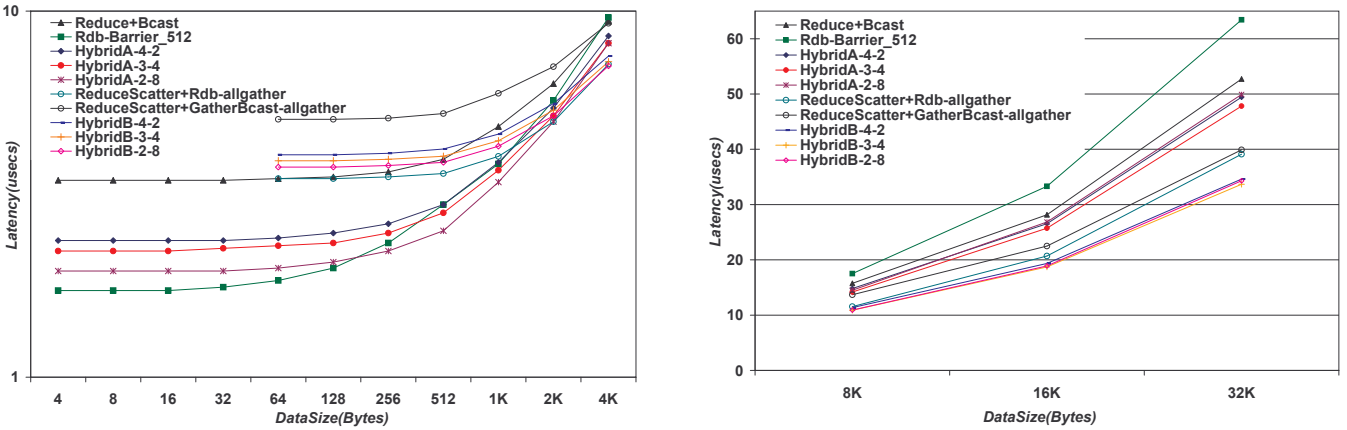
All variants of all-reduce studied here outperform CML's. They are up to 19.21 times faster than CML in the 8 SPE case and up to 9.48 times faster in the 16 SPE case. For data sizes less than 512 bytes, rdb with local store notification yields the best performance for both 8 SPE and 16 SPE systems. After that point, EIB con-



(a) 8 SPE's

(b) 16 SPE's

Figure 14: All-reduce Latency



(a) Latency for small data sizes

(b) Latency for large data sizes

Figure 15: Comparison of hybrid all-reduce with simple all-reduce algorithms.

tention becomes more important, and the barrier-enhanced rdb algorithm starts to perform the best for a few data points. In the 8 SPE case, for data sizes above 1KB, reduce-scatter plus rdb-allgather starts to perform the best. Unlike the case of reduce, where reduce-scatter improved link and computational concurrency, all-reduce already has strong concurrency in every stage because of rdb. However, reduce-scatter reduces the aggregate banwidth demand at each node by recursively halving the communication volume at each round. Rdb-allgather builds the data volume back up, but the total amount is still less than rdb-allreduce. The 16 SPE case behaves similarly, except that reduce-scatter may be followed by either rdb-allgather or gather-broadcast with similar performance for data sizes of 32 Kbytes and above. This trend for 16 SPEs arises because gather-broadcast has fewer BIF-crossing messages, motivating the design of the hybrid algorithms.

Hybrid-all-reduce latency. Figure 15 compares the new hybrid all-reduce algorithms with Rabenseifner's all-reduce algorithms and simple binomial reduce plus broadcast based algorithms. Both of these plots are for 16 SPEs, as the hybrids target inter-chip communication. Note that figure 15(a) shows the all-reduce latency for small data sizes in log-log scale while figure 15(b) shows the all-reduce latency for larger data sizes with a linear-scaled Y axis for a better view of the data points. "Rdb-Barrier_512" in the legend means the algorithm uses the barrier-enhanced rdb only for data sizes of 512 bytes or more (thus combining the best data points of the simple rdb and the barrier-enhanced rdb). Recall that *hybridA-x-y* starts as a binomial reduce, switches to rdb at round number x and has y BIF-crossing messages. HybridA-4-2 has the same number of BIF-crossing messages as reduce plus broadcast, but completes in 7 rounds rather than 8 because it collapses one round of reduce and one round of broadcast into a single round. HybridA-4-2 thus always outperforms reduce plus broadcast.

Among the all-reduce implementations, rdb performs best for small data sizes (where BIF contention is negligible). As data sizes inch upward, though, the hybrids start to outperform rdb. At a 256 byte data size, BIF contention starts to become an issue, and hybridA-2-8 is best since it has almost all of the concurrency of rdb but cuts BIF traffic in half. This trend continues until BIF

contention increases dramatically with data sizes above 2 KB. At that point, HybridB-2-8 starts to perform the best. HybridB performs the best because of the bandwidth reduction allowed by using reduce-scatter and all-gather. After 4Kb data sizes, HybridB-3-4 is the best, just as in all-gather. Our hybridA-3-4 is up to 7.3 times faster than CML's all-reduce. Note that the hybridB algorithms start to yield benefits at large data sizes (above 2 Kbytes). HybridB-3-4 is up to 15% better than Rabenseifner's algorithms and up to 11 times faster than CML.

5. RELATED WORK

Previous sections have discussed the works that are most closely related to the algorithms and results of this paper. Several papers describe implementations of collective communication operations on the Cell [17, 18, 19, 25]. Although these other approaches also strive to exploit the unique architectural features of the Cell for efficient collective communication, there are important differences between our work and the other approaches. We have compared our algorithms against more algorithms than either CML or BMM, including algorithms tuned for large data sizes that neither CML or BMM consider. They have not considered hybrid algorithms, and moreover have not considered the trade-offs between concurrency and data traffic that our hybrid algorithms manage. CML, however, was not optimized for Cell Blade collective communication, as it was designed largely for hybrid clusters like Roadrunner [6]. It is also used for transparent communication of SPEs across different nodes on a cluster.

There has also been some work on hierarchical collective communication [13, 14, 15]. The ideas that network characteristics should be exploited for efficient design is similar, but the Cell Blade is a fairly unique design and thus requires different approaches than clusters on a LAN or WAN. For example, our efficient broadcast starts with the same hierarchical design as in the work of Karonis et al. [13]. However, it then adapts the algorithm to Cell-specific constraints by optimizing the DMA ordering. Further, the previous literature does not show hybrid designs for all-gather and all-reduce as we show here.

Other architectures have special features for collective communication. IBM's BlueGene/L has multiple networks, including a torus for regular communication, a collective communication network, and a global interrupt network [2]. The collective communication network actually performs reduction operations as data packets pass through it rather than forcing the processing cores to do the computation. Although the network performs fixed-point operations, additional control software can be added at the processor cores to make it perform floating-point reductions with low latency [3]. Implementing collectives on the torus requires careful network routing to minimize hops, and the global interrupt wires can support barriers using wired-OR logic [4]. Several other works have proposed or implemented hardware support for barriers and other collective communication operations [7, 11, 23, 24]. Although the details vary, the common theme of these works is that high performance requires both hardware and software support for collective communication.

6. CONCLUSIONS AND FUTURE WORK

This paper presents new and efficient algorithms that exploit features of the Cell architecture to provide high-performance collective communication. In particular, the algorithms for broadcast, reduce, all-reduce, and all-gather are faster than any previous Cell implementations of which the authors are aware, and up to 19.21 times faster than previously-presented algorithms. Our new hybrid algorithms are faster than the previous well-known algorithms and achieve up to 35% less latency. These algorithms achieve superior performance both within a single Cell chip and across Cell chips that are part of a blade.

The superior performance of these algorithms is not simply a function of having heavily tweaked and hand-optimized implementations, but rather derives from fundamental consideration of trade-offs that arise from the multiplicity of communication mechanisms within the Cell – the overheads of these mechanisms, the effect on concurrency within collective operations resulting from the choice of communication mechanism, and the constraints of the Cell's intra-chip and inter-chip interconnects. This in turn leads to the important conclusion – applicable not just to the Cell, but also to the many complicated accelerators that are now available – that algorithmic and implementation design choices must carefully consider the global effects of decisions and how various components of the hardware and software combine to shape the performance of communication systems.

For future work, we are working on performance modeling of these algorithms for Cell-based systems. These models will serve as a guideline for developers working with collective communication on the Cell Blade and other Cell platforms. Algorithm developers can thereby focus on the design of an algorithm and determine its efficiency using the model equations rather than going through the tedious process of writing the algorithm code itself using the Cell SDK and with the architecture-specific constraints on Cell programs.

7. ACKNOWLEDGMENTS

We acknowledge Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation, for the use of Cell Broadband Engine resources that have contributed to this research. We thank IBM for providing access to their Cell blades under the VLP program. We would also like to thank Scott Pakin at LANL for helping with CML.

8. REFERENCES

[1] 30th TOP500 List, June 2008.

[2] N. R. Adiga et al. An overview of the BlueGene/L supercomputer. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pages 1–22, Nov 2002.

[3] G. Almási, G. Dózsa, C. C. Erway, and B. D. Steinmacher-Burow. Efficient implementation of Allreduce on BlueGene/L collective network. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3666 of *Lecture Notes in Computer Science*, pages 57–66. Springer Berlin, 2005.

[4] G. Almási, P. Heidelberger, C. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. D. Steinmacher-Burow, and Y. Zheng. Optimization of MPI collective communication on bluegene/l systems. In *ICS*, pages 253–262, 2005.

[5] P. Altevogt et al. Evaluating IBM Blade Center QS21 hardware performance. http://www.ibm.com/developerworks/library/pa-qs21perf/index.html.

[6] K. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho. Entering the petaflop era: The architecture and performance of roadrunner. In *IEEE/ACM Supercomputing (SC08)*, November 2008.

[7] C. J. Beckmann and C. D. Polychronopoulos. Fast barrier synchronization hardware. In *Supercomputing '90: Proceedings of the 1990 ACM/IEEE conference on*

Supercomputing, pages 180–189, Washington, DC, USA, 1990. IEEE Computer Society.

[8] J. Bruck, S. Member, C. Ho, S. Kipnis, E. Upfal, S. Member, and D. Weathersby. Efficient algorithms for all-to-all communications in multi-port message-passing systems. In *IEEE Transactions on Parallel and Distributed Systems*, pages 298–309, 1997.

[9] D. Buntinas, G. Mercier, and W. Gropp. Data transfers between processes in an SMP system: Performance study and application to MPI. *Parallel Processing, International Conference on*, 0:487–496, 2006.

[10] B. Flachs et al. PowerXCell 8i: A Cell Broadband Engine implementation enhanced for supercomputing. Presented at *HotChips 20*, August 2008.

[11] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir. The NYU Ultracomputer—designing a mimd, shared-memory parallel machine (extended abstract). *SIGARCH Computer Architecture News*, 10(3):27–42, 1982.

[12] H. P. Hofstee. Power efficient processor architecture and the Cell processor. In *HPCA*, pages 258–262, 2005.

[13] N. T. Karonis, B. R. de Supinski, I. Foster, E. Lusk, and W. Gropp. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, page 377, Washington, DC, USA, 2000. IEEE Computer Society.

[14] T. Kielmann, H. E. Bal, and S. Gorlatch. Bandwidth-efficient collective communication for clustered wide area systems. In *In Proc. International Parallel and Distributed Processing Symposium (IPDPS 2000), Cancun*, pages 492–499, 2000.

[15] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. Magpie: Mpi's collective communication operations for clustered wide area systems. *SIGPLAN Not.*, 34(8):131–140, 1999.

[16] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor interconnection network: Built for speed. *IEEE Micro*, 26(3), May-June 2006.

[17] M. Krishna et al. A synchronous mode MPI implementation on the Cell BE^{TM} architecture. In *ISPA*, pages 982–991, 2007.

[18] A. Kumar et al. A buffered-mode mpi implementation for the Cell BE^{tm} processor. In *International Conference on Computational Science (1)*, pages 603–610, 2007.

[19] S. Pakin. Receiver-initiated message passing over RDMA networks. In *22nd International Parallel and Distributed Processing Symposium (IPDPS 2008)*.

[20] F. Petrini, G. Fossum, J. Fernández, A. L. Varbanescu, M. Kistler, and M. Perrone. Multicore surprises: Lessons learned from optimizing Sweep3D on the Cell Broadband Engine. In *IPDPS*, pages 1–10, 2007.

[21] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra. Performance analysis of MPI collective operations. In *IPDPS*, 2005.

[22] R. Rabenseifner. Optimization of Collective Reduction Operations. In *Proceedings of the International Conference on Computational Science*, June 2004.

[23] L. Rudolph. Hardware support for collective communication operations. In *Parallel Architectures and Their Efficient Use*, volume 678 of *Lecture Notes in Computer Science*, pages 110–118. Springer Berlin, 1993.

[24] J. A. Test, M. Myszewski, and R. C. Swift. The Alliant FX/Series: A language driven architecture for parallel processing of dusty deck Fortran. In *PARLE Parallel Architectures and Languages Europe*, volume 258 of *Lecture Notes in Computer Science*, pages 345–356. Springer Berlin, 1987.

[25] M. K. Velamati et al. Optimization of collective communication in intra-Cell MPI. *High Performance Computing, HiPC*, 4873:488–499, 2007.