Modeling Advanced Collective Communication Algorithms on Cell-based Systems *

Qasim Ali, Samuel P. Midkiff and Vijay S. Pai

School of Electrical and Computer Engineering, Purdue University

{qali, smidkiff, vpai}@purdue.edu

Abstract

This paper presents and validates performance models for a variety of high-performance collective communication algorithms for systems with Cell processors. The systems modeled include a single Cell processor, two Cell chips on a Cell Blade, and a cluster of Cell Blades. The models extend PLogP, the well-known point-to-point performance model, by accounting for the unique hardware characteristics of the Cell (e.g., heterogeneous interconnects and DMA engines) and by applying the model to collective communication. This paper also presents a micro-benchmark suite to accurately measure the extended PLogP parameters on the Cell Blade and then uses these parameters to model different algorithms for the *barrier*, *broadcast*, *reduce*, *all-reduce*, and *all-gather* collective operations. Out of 425 total performance predictions, 398 of them see less than 10% error compared to the actual execution time and all of them see less than 15%.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent Programming

General Terms Algorithms, Performance, Measurement

Keywords Collective communication, Algorithms, Modeling

1. Introduction

Collective communication is critical in many high performance applications. Recent advances in high performance microprocessors like the Cell processor and GPUs, as well as advances in on-chip and off-chip network technology, require improved design of collective communication algorithms to effectively utilize the new architecture [4]. Such systems also require new modeling techniques for collective communication to accurately predict performance and to intelligently select among various choices of algorithms and implementations for any given operation, as many system and workload parameters can affect the performance of any particular algorithm. Among existing models, the best known and studied model for point-to-point communication is LogP, which characterizes communication performance in terms of message latency, message overhead, inter-message gap time, and the number of processes in the system [9, 10]. Extensions to LogP include PLogP, which makes the overhead and gap measures functions of the message size, and LoGPC, which includes a term for network contention [13, 15]. Other parameters that can shape the performance of a collective communication operation include the number of processes involved in the particular communication, the size of the data buffers to be communicated, the pattern of communication, and the scheduled order of data transfers.

Combining system parameters to accurately predict communication performance is difficult. The situation is worse for accelerators such as the Cell processor because of its complex and unique architecture. The on-chip Element Interconnect Bus (EIB), which is the heart of the Cell processor's communication infrastructure, can support three concurrent DMA transfers on each of its four rings. The off-chip broadband interface (BIF), which connects two Cell processors on a Cell Blade, has an order of magnitude less bandwidth than the EIB and thus can only handle a limited number of messages without being overloaded. Modeling the contention resulting from these limits is an issue for communication operations that create large numbers of messages, such as broadcast or all-reduce. Addressing these limits requires additional effort, as previous models have either ignored contention effects or considered the effect of contention as it arises on a single uniform network.

This paper makes the following contributions:

- Modifies the PLogP model by adding a contention term, and simplifies the model for Cell-based systems.
- Presents a novel suite of micro-benchmarks and a methodology to accurately determine the parameters of the extended PLogP model accounting for the unique characteristics of the Cell processor and Cell Blades.
- Introduces basic collective communication patterns which are used as building blocks for various collective communication algorithms and develops models for them, in the context of the Cell processor and its hardware features, using the parameters determined above.
- Uses the above parameters to model advanced and efficient collective communication algorithms for a single Cell processor and the Cell Blade.
- Shows that the models can easily be plugged into existing cluster-based models for collective communication

We claim and show that performance modeling of an individual Cell processor and a Cell blade is different from other systems because of their novel architectures. Generic models do not work in the aforementioned systems as we show in our experimental section. We expand the abstract PlogP model to account for architecture-specific details of the Cell. We also show

*This work is supported in part by the National Science Foundation under Grant Nos. CCF-0325603, CNS-0509390, CCF-0532448 and CNS-0751153, CCF-0916901, CCF-0833115, CNS-0707931.

PPoPP'10, January 9–14, 2010, Bangalore, India.

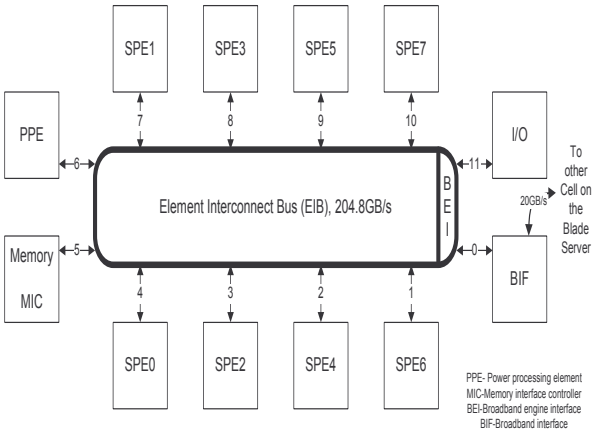

Figure 1: Cell architecture overview.

how this modeling can be extended to a cluster of Cell Blades or Roadrunner-like systems by adapting the cluster-level models and techniques previously proposed by Dongarra et al. and Kielmann et al. [13, 17].

Empirical data shows that our models perform well: 94% of our 425 performance predictions show errors of less than 10% and all of the predictions show errors less than 15%. The errors are small enough that these predictions can be used to make correct decisions about which algorithm to choose for a particular operation before implementing the operation. This capability is particularly valuable for systems, such as the Cell, that require architecture-specific programming efforts. In contrast, using previous non-Cell-specific methods to capture the inter-message gap parameter or ignoring BIF contention would lead to large and highly variable errors, ranging from no difference at all up to 140% error. Consequently, these predictions cannot guide proper algorithm choices, further validating the need for a new model and new methods to capture the model parameters.

2. Cell Architecture and Communication Overview

This section describes the Cell and features that are germane to the design and modeling of efficient collective communication. Figure 1 gives a simplified illustration of the Cell architecture. The Cell processor is a heterogeneous multicore chip consisting of one general-purpose 64-bit Power processor element (PPE), eight specialized SIMD coprocessors called synergistic processor elements (SPEs), a high-speed memory controller, and a high-bandwidth bus interface, all integrated on-chip. Each SPE consists of a synergistic processor unit (SPU) and a memory flow controller (MFC). The MFC includes a DMA controller, a memory management unit (MMU) to allow the SPEs to use virtual addresses when performing DMAs, a bus interface unit, and an atomic unit for synchronization with other SPEs and the PPE. Each SPU includes a 256 Kbyte local-store (LS) memory to hold its program's instructions and data; the SPU does not have any hardware-managed cache. The SPU cannot access main memory directly, but it can issue DMA commands to the MFC to bring data into local store or write computation results to main memory.

The PPE and 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 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 supports only a non-coherent I/O interface (IOIF) protocol, 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. More details about the Cell can be found in [14].

3. Performance Modeling for the Cell

Performance models are a basis for the design and analysis of parallel algorithms. A good model includes a small number of parameters but can express the complexity of the underlying runtime and hardware across a wide range of different scenarios. Since the collective communication algorithms used are based on point-to-point messages, we extend the standard PLogP model to model the Cell-targeted collective communication algorithms we previously presented by carefully analyzing the communication characteristics of each algorithm [4]. Since some of the collectives have both a communication and computation part (e.g., all-reduce), we model both the network and computation aspects of the Cell system.

3.1 The PLogP model and our extensions

To model the communication part of a collective, we extend the popular PLogP model [13]. PLogP characterizes the network in terms of latency L, send and receive overheads o_s and o_r, gap required between messages g, and the number of nodes (processors) in the system, p. The latency, gap and overheads are dependent on message size. In PLogP, the time to send a message between two nodes is given by $L + g(m)$.

As in other collective communication models (e.g., Dongarra et al. [17]), we modify the p parameter to represent the number of processors *in a particular communication*. The algorithms we model only use SPE-to-SPE communication, so p is the number of SPEs in the collective. Following the LoGPC model, we add a congestion/contention term, C [15]. In our model, this is used only to represent contention on the lower-bandwidth BIF and the methodology to measure it is different than LoGPC. Our C parameter depends on the message size m and number of messages crossing the BIF n. The parameters of our model are summarized as below:

- p: the number of SPEs participating in a collective opertaion;
- L: Communication delay (upper bound on the latency with no contention from LS of one SPE to LS of another SPE);

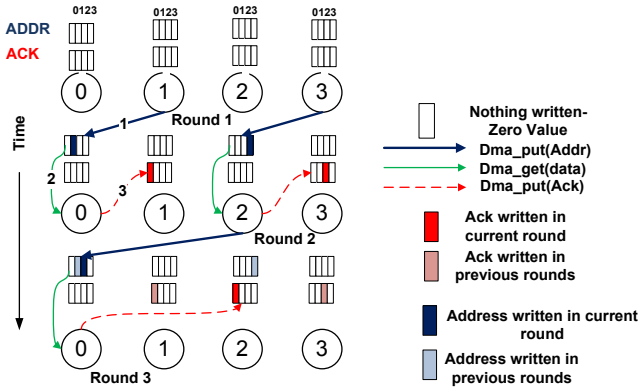

Figure 2: Local store based communication and synchronization for simple binomial tree based reduce.

- g : Gap (indirect communication bandwidth, minimum interval between consecutive messages; bandwidth $\sim \frac{1}{g}$);
- C: Contention/Congestion (link contention on the BIF).

We model the computation time for a message of size m as γm, where γ is computation time per byte.

3.2 Basic communication structure

Almost all of the collective communication algorithms that we study 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. The SPEs maintain notification arrays in their local stores, where each notification array has x entries, with x being the number of SPEs participating in a particular collective. Each element of a notification array is 16 bytes long. These arrays are used for address (ADDR) communication and acknowledgments (ACKs) between the SPEs in a lock-free manner [8]. Figure 2 shows the working of these for a simple binomial-tree based reduce. An SPE first does a dma_put of the address of the data to be communicated. Then the receiver does a dma_get of the data, and when the data is received it does a dma_put of an ACK into the sender's local store.

3.3 Formulation and measurement of the extended PLogP model parameters on the Cell Blade

Ainsworth and Pinkston formulated the latency of packets across the EIB as $Latency = Send\ Phase + Cmd\ Phase + Data\ Phase + Recv\ Phase$ [2]. The sending phase primarily consists of setting up the DMA transfer. The command phase consists of command issue and snoop responses. Both of these are independent of the message size. Hence the sender overhead is constant. The data phase is the actual transit time of the DMA transfer. Finally, in the receiving phase, the data is moved from the bus interface unit to the local store memory. The receiving phase is part of the DMA transfer if IBM's SDK3.0 library is used. Hence receiver overhead can be assumed to be zero. Also because of the nature of the communication structure used, the time to send a message between two nodes is $L(m)$ (with overheads inclusive), because there is no $g(m)$. $g(m)$ appears only when an SPE tries to send a message before the first one gets completed.

Using micro-benchmarks, we measure three model parameters for the Cell processor: latency, gap and congestion, all of which are message size dependent. Other parameters such as the sender and receiver overhead are assumed to be constant and independent of the message size [2]. Completely ignoring o is not feasible for algorithms that use small messages (e.g., barrier collective), but we combine o with L to keep the models simple.

Sender:
dma_put(ACK);
while(wait for ACK from receiver);
Receiver:
while(wait for ACK from sender);
dma_put(ACK);

(a) RTT with ACK array.

Sender:
dma_put(ADDR);
while(wait for ADDR from receiver);
dma_get(DATA, ADDR);
Receiver:
while(wait for ADDR from sender);
dma_get(DATA, ADDR);
dma_put(ADDR);

(b) RTT with ADDR array.

Figure 3: Latency micro-benchmarks.

L **parameter micro-benchmark.** We define $L(m)$ to be half of the round trip time (RTT) of a message of size m. We use two micro-benchmarks to measure $L(m)$. The latency from the first benchmark is used to model synchronization algorithms that do not transfer data; hence, they only use one synchronization array called the ACK array. The second micro-benchmark is used to model algorithms which transfer data and synchronize. These algorithms use both ADDR and ACK arrays. As indicated in Section 3.2, in order to transfer data the sender first sends the address of the data via dma_put, and the receiver gets it via dma_get. In the first micro-benchmark, shown in Figure 3(a), the sender directly sends an ACK message to the receiver's local store. When the receiver gets the ACK, it returns an ACK. In the second micro-benchmark (Figure 3(b)), the sender first sends the address of the data to be sent into the receiver's local store via dma_put. Then the receiver issues a dma_get to get the data for the specified address and sends the address of its data to the sender. Finally the sender also issues a dma_get to get the data. Both micro-benchmarks measure the RTT of a message of size m. RTT is divided by two to get $L(m)$.

g **parameter micro-benchmark.** The g parameter is calculated in a completely different way than Kielmann et al. [13] do, which is a standard way to compute g in the MPI community and used by many, e.g., Dongarra et al. [17]. Ideally, the g parameter would be measured as follows. Consider a number of SPEs that issue a dma_get command. The first dma command processed by the data arbiter would be scheduled to do a data transfer. The time that the second SPE dma command must wait to start the data transfer would be the value of the g parameter.

Unfortunately, g cannot be measured like this on the Cell. In order to understand why, consider the basic flow of a DMA transfer [14]. The SPU first issues a DMA command queue which is processed by the DMA controller. Next the DMA controller creates a bus request to transfer the next block of data for the command. This bus request can transfer up to 128 bytes of data. The controller then queues this bus request to the Bus interface unit (BIU). The BIU selects the request from its queue and issues the data transfer command to the EIB. The EIB consists of four 16-byte wide data rings and the EIB's data arbiter implements a round robin bus arbitration. As all SPEs can issue a dma_get command at the same time, each of them will be posting requests for 128 bytes of data (if the total

requested data is equal to or more than 128 bytes, and less than 128 bytes if the requested data is less).

Because we cannot change the data arbiter policy, we employ the following technique to measure the gap parameter. We designate one SPE as the root which has the data. All other (non-root) SPEs issue a dma_get of data from this root SPE. The time taken by each SPE is measured, and these times are averaged over the number of SPEs issuing the dma_get command. The value of g obtained in this manner would be similar to what the value of g would be if we changed the data arbiter policy to "first-come, first-served" and let the bus requests issued by one SPE finish before the bus request of another SPE is serviced. For two Cell processors on a blade, there are two g parameters, the intra-chip g_1 and inter-chip g_2. We only use the single g when a single Cell processor is involved in the communication.

C parameter micro-benchmark. The congestion parameter is calculated by the butterfly communication pattern micro-benchmark, i.e., each SPE sends and receives data from another distinct SPE. When two SPEs are engaged in the data transfer (both sending and receiving) across the BIF, we denote this as BIF-2. Thus, BIF-16 means 16 messages are crossing the BIF. BIF-2 latency is the average of the time taken by the two SPE DMAs. We use BIF-2 as the baseline since we have observed that BIF-1 equals BIF-2; that is, the BIF can support two simultaneous messages without any observable contention. We define the C parameter, for a given message size m and number of BIF-crossing messages n, as the excess latency observed by a BIF-n butterfly communication pattern when compared to a BIF-2 pattern. As the values of n and m increase, the congestion also increases.

Some other notation used in our models is $L_1(1)$ and $L_1(m)$, which are the intra-chip latencies. The "1" inside the parentheses indicates that the message size is a small constant; more specifically, it is 16 bytes long as this is the smallest DMA transfer possible on Cell and m is the message size in bytes. Similarly $L_2(1)$ and $L_2(m)$ are latencies for inter-chip transfers. g_{Ack} is used to denote the gap for ACKs.

4. Building Blocks for Collective Communication Models

This section presents an overview of the three basic collective communication patterns that serve as building blocks for the algorithms used in this paper. We will briefly explain these patterns and develop models for them in the context of the Cell processor and its DMA engine. In Figure 4 the numbers in circles represent nodes, and the arrows (both dotted and dark) represent communication.

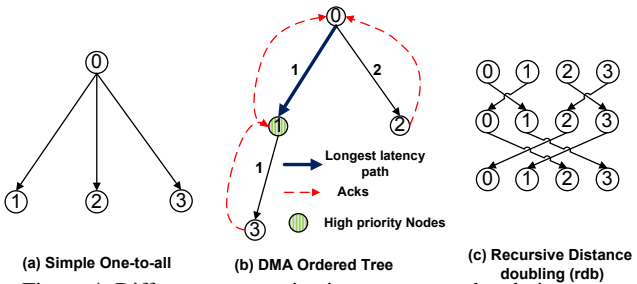

Figure 4: Different communication patterns and techniques.

Figure 4(a) shows a one-to-all (OTA) communication pattern in which all nodes receive data from just one node. If all the nodes simultaneously do a dma_get, then a communication gap will arise that depends on g. Let q be the number of inter-message gaps (g) required. This value is based on the maximum out-degree of a node at each level of the tree and is given by the following equation:

$$q = \sum_{k=1}^{h} (d_k - 1)$$

where d_k is the maximum out-degree of a node at level k and h is the height of the tree. In the example of Figure 4(a), h is 1 and $d_1 = 3$, so there will be two gaps ($q = 2$).

Similarly, in the case of inter-chip SPE communication, two q's will be used, q_1 for the number of g_1 (on-chip gap) terms and q_2 for the number g_2 (inter-chip gap) terms given below:

$$q_1 = \sum_{k=1}^{h} (d_k - n_{k2} - 1)$$

$$q_2 = \sum_{k=1}^{h} (d_k - n_{k1} - 1)$$

where n_{k1} and n_{k2} are the number of messages at level k between chips and across chips, respectively.

The second pattern is the optimized ordered tree pattern shown in Figure 4(b). The gap parameter which arises in Figure 4(a) can be eliminated by inserting ACKs to order the DMAs. The ACKs are depicted in Figure 4 (b). To do this, the root node first does a dma_put into the dark colored node, because it is on the longest latency path, and then into the white colored node. The cost of the communication pattern shown in Figure 4(b) is $2 \times (L(m) + L(1))$. If the DMAs were not ordered the cost would have been $2 \times L(m) + g(m)$: one g term in the first level and no g term in the second level of the tree.

The third pattern shown in Figure 4(c) is based on an efficient algorithm known as recursive distance doubling or the butterfly algorithm [19]. Figure 4(c) shows the butterfly algorithm in action. Each node sends to, and receives from, a different partner node at each stage of the algorithm: first with distance 1, then with distance 2, then 4, and so forth. At the end of the last stage all nodes have received (reduction and gather operations) and, if necessary, processed the data (reduction operations). When applied to all-reduce, this algorithm specifically exploits full-duplex communication links to collapse a reduce and broadcast into a single operation. Each round of the butterfly pattern requires $L(m)+L(1)$ time (data plus ACK) based on the interconnect parameters. If, however, the BIF is the network used at any stage, sending too many messages across it may lead to poor performance. In particular, if the number of BIF messages is greater than two and the data size is greater than 256 bytes, the network starts to become congested and becomes a bottleneck. We model the congestion on the BIF by a parameter C. These thresholds were determined experimentally.

If the number of SPEs involved is not a power of 2, the first stage communicates from the higher-numbered SPEs to the lower-numbered SPEs within the next lower power of 2 [18]. The SPEs within the next lower power of 2 proceed using the butterfly algorithm. After that, the lower-numbered SPEs feed information back to the higher-numbered SPEs, for a total of $2 + \lfloor \log_2 p \rfloor$ steps for p numbers of SPEs in the all-reduce.

5. Modeling Collective Communication Algorithms

Using the communication flow of each algorithm, the point-to-point messages modeled using the standard PLogP parameters, our modeling extensions described in Section 3 and the three basic communication patterns discussed and modeled in Section 4, we have developed models for different algorithms for the barrier, reduce, all-reduce, broadcast and all-gather collectives. All the algo-

rithms used in this paper are based on inter-SPE communication and the data to be communicated resides in the local store, similar to the Cell Messaging Library, which is an MPI implementation for the Cell [16]. We will explain a few representative algorithms to explain how the models are developed. The details of all the algorithms can be found in our previous work [4].

For one-to-all (OTA) models (barrier and broadcast), the root node is assumed to be on the first chip, without loss of generality.

5.1 Barrier models

We use the results of the micro-benchmark in Figure 3(a) to model the barrier algorithms because they transfer no message data and only use an ACK array. Note that if more than eight SPEs are involved in a particular collective, the first eight SPEs are on the first Cell processor and the rest are on the second. This arrangement minimizes the number of BIF crossings. The cost of recursive doubling follows from Section 4, except that the actual cost of a recursive doubling round is only $L(1)$, not $L(m) + L(1)$, because there is no data transfer. In Bruck, step k of the collective has an SPE of rank r sending a message to an SPE of rank (r+2^k) and receiving a message from an SPE of rank (r-2^k) with wrap around [7, 17]. Bruck always needs $\lceil \log_2 p \rceil$ rounds for p nodes. In Bruck, 2^{round_num} inter-chip messages are exchanged at each round, leading to higher latencies in higher numbered rounds. $round\_num$ is the total number of communication rounds/phases. As there are at least two messages crossing the BIF in each round, the overall latency in each round is $max(L_1(1), L_2(1))$, i.e., the maximum of the inter-chip and intra-chip latency. Because there are $\lceil \log_2 p \rceil$ steps, the overall communication time for two chips is modeled as $\lceil \log_2 p \rceil \times (max[L_1(1), L_2(1)])$.

In a one-to-all barrier, the root node sends $(p-1)$ messages. The first message reaches the first non-root SPE in time $L(1)$, the second message arrives in time $g(1)$ because it has to wait $g(1)$ amount of time, and so on. Thus it takes a total of $g(1) \times (p-2) + L(1)$ time to broadcast a message to $p-1$ SPEs, thus the value of q is $p-2$. After all $p-1$ SPEs get their messages, they send an ACK back to the root node. In this phase, the time will be at least $L(1)$ plus some gap experienced by the ACKs. The total gap is given by $0 \leq g \leq ((p-2) \times g_{Ack}(1))$. Some of these ACKs will be overlapped with the original messages in the first phase. Other barrier algorithms were modeled similarly and are shown in Table 1.

5.2 Broadcast models

We use the second latency micro-benchmark shown in Figure 3 to model different broadcast algorithms and the other algorithms presented in the remainder of this paper. The "simple" suffix in Table 2 denotes a broadcast in which nodes do not wait for ACKs at the end of each communication round, allowing later communication rounds to complete their data transfers before earlier ones.In the simple version of the binomial tree algorithm, the root node sends the address of the data to be broadcast to all of its children and they issue a dma_get at approximately the same time. Because the DMA requests are 128 bytes in length, each child will get 128 bytes of data initially rather than SPE 1 getting all the data. The result of this is that SPE 1 will get the data after $L(m) + q \times g(m)$ time, not $L(m)$ time. This in turn delays the children of SPE 1. Hence at each level there is an additional delay, which we model by $g(m)$. We call such behavior a priority inversion.

We refer to an optimized version of broadcast that prevents priority inversion of messages by ordering DMAs as "opt". This is the second communication pattern discussed in Section 4. To avoid the priority inversion problem in the simple broadcast, our optimized broadcast uses address-data-ACK phases at each round as described in Section 3.2. Each round takes $L(m)+L(1)$ time. By using such ordered DMAs, the overall latency will not have $g(m)$ terms in the case of a power-of-two number of SPEs.

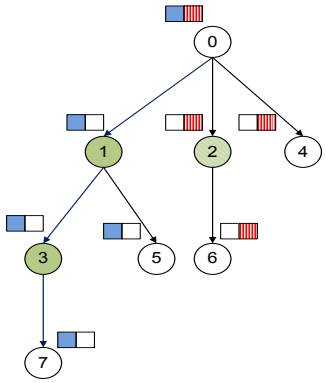

Figure 5: First phase of the segmented binomial algorithm.

A segmented binomial tree has a two-phase broadcast process. In the first phase, the nodes in the left half of the tree get the left half of the data and the nodes in the right half of the tree get the right half. This step takes $(\log_2 p - 1) \times L(m/2) + L(1)$ time. In the second phase, the left and right half nodes exchange their data, which takes $L(m/2) + L(1)$ time. (All of our segmented trees assume the DMA ordering optimization.) In Figure 5, the left half of the data is shown as a dark color (blue) and the right half is shown as a light color (red). In the 16 SPE case, this policy results in 14 inter-chip data exchanges in the last phase, causing substantial congestion when larger messages are used. This is shown by the $C(m > 256)$ term in the segmented-binomial model since sending more than two large messages across the BIF will result in some congestion and delay, which is accurately captured by our congestion micro-benchmark.

To reduce the number of BIF-crossing messages in the 16 SPE case, we implemented a hybrid version of the segmented binomial, where the root node sends all data to one representative node in the second chip. This takes $L_2(m)$ for the data message and $L_2(1)$ for the ACK, after which the two-phase broadcast times are all on-chip and thus based on L_1.

5.3 All-gather Models

In recursive distance doubling for all-gather, p messages are exchanged in each communication round and the data being communicated doubles at each stage. For the two chip scenario, in the first $\log_2 p - 1$ rounds, data communication will take place on the EIB and the communication cost in each round is $L_1(messagesize) + L_1(1)$. In the last round, all the data traffic will be on the BIF. Because the number of messages crossing the BIF interface is more than two, the congestion C term is needed in the final equation shown in Table 3.

Consider the hybrid algorithm called hybrid-x-y (shown in Figure 6) that seeks to find an ideal compromise between the concurrency of recursive doubling and the low traffic of gather followed by broadcast. x is the round number where the switch to recursive doubling occurs and y is the number of data messages crossing the BIF. For simplicity, consider the case of p nodes where p is a power of 2. This algorithm starts by performing $x-1$ rounds of binomial gather, leaving $\frac{p}{2^{x-1}}$ evenly-spaced node IDs in the communication structure. The cost of each step is $L_1(messagesize) + L_1(1)$, where the message size depends on the round number, and is $2^{round\_number} \times messagesize$ bytes. At this point, the algorithm switches to recursive doubling until all $\frac{p}{2^{x-1}}$ nodes have the results of the all-gather. This takes $\log_2 p - (x-1)$ rounds and results in $\frac{p}{2^{x-1}}$ messages across the BIF when the lower half and upper half of the nodes are on separate chips. The cost of each

Barrier	One/Two Cells	Model equation
Recursive doubling	One	T = $\log_2 p \times L(1)$, p is a power-of-two T = $(\lfloor \log_2 p \rfloor + 2) \times L(1)$, otherwise
Recursive doubling	Two	T = $(\log_2 p - 1) \times L_1(1) + L_2(1)$, p is a power-of-two T = $(\lfloor \log_2 p \rfloor + 1) \times L_1(1) + L_2(1)$, otherwise
Bruck	One	T = $\lceil \log_2 p \rceil \times L(1)$
Bruck	Two	T = $\lceil \log_2 p \rceil \times (max[L_1(1), L_2(1)])$
Gather-Broadcast	One	T = $2 \times \lceil \log_2 p \rceil \times L(1)$
Gather-Broadcast	Two	T = $max[2 \times (\lceil \log_2 p \rceil - 1) \times L_1(1),$ $2 \times [L_2(1) + (\lceil \log_2 p \rceil - 1) \times L_1(1)]]$
One-to-all	One	T = $q \times g(1) + L(1) + L(1) + X,$ $0 \leq X \leq (q \times g_{Ack}(1))$ and $q = p - 2$
One-to-all	Two	T = $2 \times (L_1(1) + L_2(1)) + q_1 \times g_1(1) + q_2 \times g_2(1) + X,$ $0 \leq X \leq q_1 \times g_{1Ack}(1) + q_2 \times g_{2Ack}(1)$ and $q_1 = 6, q_2 = p - 9$

Table 1: Models of different Barrier algorithms.

Broadcast	One/Two Cells	Model equation
One-to-all	One	T = $q \times g(m) + L(m) + L(1) + X,$ $0 \leq X \leq (q \times g_{Ack}(1))$ and $q = p - 2$
One-to-all	Two	T = $L_1(m) + q_1 \times g_1(m) + L_2(m) + q_2 \times g_2(m)$ $+L_1(1) + L_2(1) + X, 0 \leq X \leq$ $q_1 \times g_{1Ack}(1) + q_2 \times g_{2Ack}(1), p > 8$ and $q_1 = 6, q_2 = p - 9$
Binomial-Tree-simple	One	T = $\lfloor \log_2 p \rfloor \times [L(m) + L(1)] + q$ $\times g(m) + X, 0 \leq X \leq q \times g_{Ack}(1))$, and $q = 3$
Binomial-Tree-opt	One	T = $\log_2 p \times [L(m) + L(1)]$, where p is power-of-two
Binomial-Tree-simple	Two	T = $(\lfloor \log_2 p \rfloor - 1) \times [L_1(m) + L_1(1)] + L_2(m) + L_2(1)$ $+q_1 \times g_1(m) + X, 0 \leq X \leq q$ $\times g_{1Ack}(1))$, and $q_1 = 7, q_2 = 0$
Binomial-Tree-opt	Two	T = $(\lceil \log_2 p \rceil - 1) \times [L_1(m) + L_1(1)] + L_2(m) + L_2(1)$
Segmented-Binomial-opt	One	T = $(\log_2 p + 1) \times [L(m/2) + L(1)]$, p is a power-of-two
Segmented-Binomial-opt	Two	T = $(\log_2 p - 1) \times [L_1(m/2) + L_1(1)] +$ $2 \times [L_2(m/2) + L_2(1)] + C(m/2 > 256)$, p is a power-of-two
Hybrid-Seg-Binomial-opt	Two	T = $\log_2 p \times [L_1(m/2) + L_1(1)] + [L_2(m) + L_2(1)]$, p is a power-of-two

Table 2: Models of different Broadcast algorithms.

round would be the same as above, except that the cost should be $L_2(messagesize)+L_2(1)$ for rounds that cross the BIF. A C term is added if the messages crossing the BIF are large and more than two in number. Now, each of the $\frac{p}{2^{x-1}}$ nodes with the result act as the root of binomial broadcast trees, requiring $x-1$ more rounds to give the result to all nodes. The cost of each of these steps is also $L_1(messagesize) + L_1(1)$. Thus, the total number of rounds are $\log_2 p + (x-1)$ and there are $\frac{p}{2^{x-1}}$ BIF-crossing data messages. In Table 3, m is the starting size of the send buffer and $p \cdot m$ is the size of the final receive buffer.

5.4 Reduce Models

The binomial reduce is the opposite of binomial broadcast. In this case, however, there is a computation term γ included in the models. For large data sizes, we studied Rabenseifner's reduce, which is based on reduce-scatter and gather [19]. Reduce-scatter effectively decreases the aggregate bandwidth requirement by giving each node only a subset of the reduced data. The amount of data to be communicated is halved at each stage along a binomial tree and computation is performed on the communicated data. The cost of each stage k is $L(m_k) + \gamma m_k + L(1)$, where the message size m_k depends on the round number k as $(2^{log_2 p-k})$. In the gather phase, the amount of data is doubled at each stage, and the root node contains the total reduced data, with the cost being $L(messagesize) + L(1)$ for each stage. Thus the resulting overall time for reduce-scatter plus gather based reduction is $\sum_{k=0}^{\log_2 p-1}[2L(2^k m) + \gamma 2^k m + 2L(1)]$. For 16 SPEs, the congestion (C) term is added since in the first round of reduce-scatter there will be 16 messages crossing the BIF. In Table 4, the reduce-scatter model assumes that the total size of the reduced data is $p \cdot m$, with m being the size of the data at each node when the reduce-scatter is done.

5.5 All-reduce Models

The Reduce-scatter plus recursive distance doubling all-gather is similar to reduce-scatter plus gather-based reduce with the addition of another congestion term (C). When 16 SPEs are participating in the all-reduce, the reduce-scatter phase has 16 messages crossing the BIF and so does the all-gather phase.

HybridA-3-4 switches to recursive doubling in the third communication round ($x - 1 = 2$). HybridA-3-4 starts with $x - 1 = 2$ rounds of binomial reduce with the cost of each round being $L_1(m) + L_1(1) + \gamma \cdot m$ and then switches to recursive doubling all-reduce in round $x = 3$. The number of data messages crossing the BIF is 4. The cost of each round is the same as above, but $L_2(messagesize) + L_2(1)$, if $x > 3$ and a C term added

All-gather	One/Two Cells	Model equation (p is a power-of-two)
Recursive doubling	One	T = $\sum_{k=0}^{\log_2 p-1}[L(2^k m) + L(1)]$
Recursive doubling	Two	T = $\sum_{k=0}^{\log_2 p-2}[L_1(2^k m) + L_1(1)]$ $+L_2(2^{\log_2 p-1} m) + L_2(1) + C(2^{\log_2 p-1} m > 256)$
Hybrid-x-y	Two	T = $\sum_{k=1}^{x-1}[L_1(2^{k-1} m) + L_1(1)]+$ $\sum_{k=x}^{\log_2 p}[L_z(2^{k-1} m) + L_z(1) + C(2^{k-1} m > 256$ and $y > 2$ and $z = 2)]$ $+(x-1)[L_1(p \cdot m) + L_1(1)]$, where z =2, if $k > 3$ and z = 1 otherwise

Table 3: Models of different All-gather algorithms.

Reduce	One/Two Cells	Model equation
Binomial	One	T = $\lfloor \log_2 p \rfloor \times [L(m) + L(1) + \gamma m]$
Binomial	Two	T = $(\lceil \log_2 p \rceil - 1) \times [L_1(m) + L_1(1) + \gamma m]+$ $L_2(m) + L_2(1) + \gamma m$
ReduceScatter+gather	One	T = $\sum_{k=0}^{\log_2 p-1}[2 \times L(2^k m) + \gamma 2^k m + 2 \times L(1)]$, p is a power-of-two
ReduceScatter+gather	Two	T = $\sum_{k=0}^{\log_2 p-2}[2 \times L_1(2^k m) + \gamma 2^k m + 2 \times L_1(1)]$ $+2 \times [L_2(2^{\log_2 p-1} m) + L_2(1)] + \gamma 2^{\log_2 p-1} m$ $+C(2^{\log_2 p-1} m > 256)$, p is a power-of-two

Table 4: Models of different Reduce algorithms.

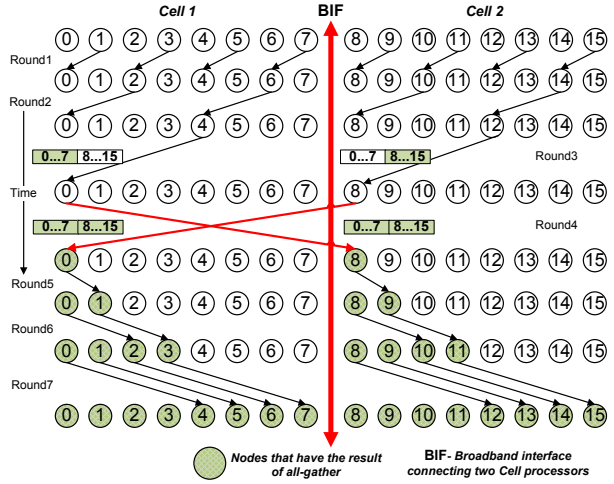

Figure 6: Hybrid 4-2 all-gather (switching round=4, data messages across BIF=2).

when the number of messages crossing the BIF is greater than 2. At the end of fourth round, SPEs 0, 4, 8 and 12, will have the results of the all-reduce, and each of them will act as the root of a four node binomial-tree. Each round of this four-node binomial-tree takes $L_1(m) + L_1(1)$ time and 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. In HybridA-x-y all-reduce, the binomial reduce takes $x - 1$ rounds, the recursive doubling all-reduce takes $\log_2 p - (x - 1)$ rounds and the binomial broadcast takes another $x - 1$ rounds.

The HybridB algorithm first uses Rabenseifner's reduce-scatter to reduce traffic. Then, HybridB invokes the hybrid all-gather algorithm discussed in Section 5.3. HybridB-x-y means that it uses hybrid-x-y all-gather.

6. Extending the Model to a Cluster of Cell-based Systems

In this section we show Our performance models can be extended to a cluster of Cell based systems and complex hybrid systems like the Roadrunner [6].

6.1 Model extension for a cluster of Cell Blades

Let $L_{spe-ppe}(m)$ be the latency from an SPE to a PPE or vice versa on a Cell Blade, and $g_{spe-ppe}(1)$ be the ACK from an SPE to the PPE and vice versa. These parameters can be measured using benchmarks similar to those used to measure inter-SPE parameters. Define $L_{node}(m)$ and $g_{node}(m)$ as the latency and gap from one Cell Blade to another Cell Blade (i.e., a different node). We measure these parameters using the traditional PLogP micro-benchmarks [10, 13] using the logp_mpi software. We used MPICH2-1.1 for inter-node communication [1].

Cluster Barrier Implementation and Modeling. Consider a hybrid barrier implementation for N Cell Blades. We use a recursive doubling barrier implementation on the individual Cell Blades. The cost of 16 SPEs synchronizing on one Cell Blade is $3 \cdot L_1(1) + L_2(1)$. One PPE on each Blade will receive ACKs from 16 local SPEs; the cost of this will be the latency from SPE to PPE and the gap, $L_{spe-ppe}(1) + g_{spe-ppe}(1)$. There will be some overlap in the above two phases and the overall cost will be the maximum of the two. After receiving ACKs, the PPEs on each Cell Blade will participate in an MPI Barrier. The MPICH2-1.1 uses Bruck for their barrier implementation. The cost of this would be $\lceil \log_2 N \rceil [L_{node}(1) + g_{node}(1)]$. There will be some overlap in the inter-SPE recursive doubling on the SPEs and the inter-node Bruck barrier cost. Hence the cost will be the maximum of these two. When each Cell Blade (one PPE) is finished with the Bruck Barrier, it will send an ACK to one other SPE and that SPE will send an ACK to other SPEs in a binomial fashion. The cost of this step would be $L_{ppe-spe}(1) + 3 \cdot L_1(1) + L_2(1)$.

As in this example for barriers, the performance models of other collectives could also be extended to work on cluster of Cell Blades.

All-reduce	One/Two Cells	Model equation (p is power-of-two)
Recursive doubling	One	T = $\log_2 p \times [L(m) + 2 \times L(1) + \gamma m]$
Recursive doubling	Two	T = $(\log_2 p - 1) \times [L_1(m) + 2 \times L_1(1) + \gamma m] +$ $L_2(m) + 2 \times L_2(1) + \gamma m + C(m > 256)$
ReduceScatter+rd-allgather	One	T = $\sum_{k=0}^{\log_2 p-1} [2 \times L(2^k m) + \gamma 2^k m + 2 \times L(1)]$
ReduceScatter+rd-allgather	Two	T = $\sum_{k=0}^{\log_2 p-2} [2 \times L_1(2^k m) + \gamma 2^k m + 2 \times L_1(1)]$ $+2 \times [L_2(2^{\log_2 p-1} m) + L_2(1)] + \gamma 2^{\log_2 p-1} m$ $+2 \times C(2^{\log_2 p-1} m > 256)$
ReduceScatter+gather+Bcast	One	T = $\sum_{k=0}^{\log_2 p-1} [2 \times L(2^k m) +$ $\gamma 2^k m + 2 \times L(1)] + \log_2 p \times [L(m) + L(1)]$
ReduceScatter+gather+Bcast	Two	T = $\sum_{k=0}^{\log_2 p-2} [2 \times L_1(2^k m)$ $+\gamma 2^k m + 2 \times L_1(1)]$ $+2 \times [L_2(2^{\log_2 p-1} m) + L_2(1)] + \gamma 2^{\log_2 p-1} m +$ $C(2^{\log_2 p-1} m > 256) + (\log_2 p - 1) \times [L_1(m) + L_1(1)] +$ $L_2(m) + L_2(1)$
HybridA-x-y	Two	T = $2 \times (x-1)[L_1(m) + L_1(1)] + (x-1) \times \gamma m$ $\sum_{k=x}^{\log_2 p} [L_z(m) + L_z(1) +$ $\gamma m + C(m > 256$ and $y > 2$ and $z = 2)]$, where z =2, if $k > 3$ and z = 1 otherwise
HybridB-x-y	Two	T = $\sum_{k=0}^{\log_2 p-2} [L_1(2^k m) + \gamma 2^k m + L_1(1)]$ $+L_2(2^{\log_2 p-1} m) + L_2(1) + \gamma 2^{\log_2 p-1} m$ $C(2^{\log_2 p-1} m > 256)$ + *Hybridall-gather-x-y*

Table 5: Models of different All-reduce algorithms.

6.2 Model extension for Cell-based hybrid systems

Similarly these models can be extended to work on hybrid systems like the Roadrunner, by accounting for the additional levels of hierarchy and interconnect. Roadrunner has a deep communication hierarchy, including an Element Interconnect Bus (EIB), BIF (Broadband interface), PCI Express, Hypertransport, and Infiniband. Each of these interconnects has a different latency. Define $L_3(m)$ and $g_3(m)$ as the latency and gap from a Cell blade to an Opteron blade on the triblade Roadrunner node (over PCIe) and $L_4(m)$ and $g_4(m)$ as the latency and gap from one Opteron to another Opteron (different node). Note that the third and fourth level parameters could be benchmarked using the traditional LogP/PlogP measures as these are more appropriate for conventional networks.

Roadrunner Barrier Modeling. Consider a recursive doubling barrier for N triblade nodes. The cost of 16 SPEs synchronizing on one Cell Blade is $3 \cdot L_1(1) + L_2(1)$. One PPE on each Blade will receive ACKs from 16 local SPEs; the cost of this will be the latency from SPE to PPE and 15 times the gap. There will be some overlap in the above two phases and the overall cost will be the maximum of the two. After receiving ACKs, the two PPEs will send an ACK to the Opteron in approximately $L_3(1)$ time, because each Cell blade has an independent PCI Express link. Finally the Opterons will engage in the global recursive doubling phase and this step will take $\log_2 N$ steps when N is a power of two. Each step will have a cost of $L_4(1)$. When each Opteron blade is be done with recursive doubling, it will send ACKs to two PPEs (one per Cell blade), costing another $L_3(1) + g_3(1)$. Each PPE will then forward the ACK to one representative SPE and that SPE will do a one-to-all broadcast or binomial broadcast of the ACKs to all other SPEs. The model for this is already covered in the previous section 6.1. As in this example for barriers, the performance models of other collectives can be extended to account for the additional levels of hierarchy and interconnect in the Roadrunner.

7. Experimental Evaluation

All experiments reported here were performed on the IBM BladeCenters QS20 and QS22 at Georgia Tech. The QS2x organizes two 3.2Ghz Cell processors into an SMP configuration, connected by the BIF inter-chip interconnect. All code shown here uses IBM's SDK3.0 and were compiled using 64-bit gcc with optimization level `-O3`. Timings are measured using the fine-grained decrementing register provided by the Cell blade. All latency numbers were averaged over 100,000 iterations. These results are compared with the predictions of the performance models of Section 5.

7.1 Model parameters.

We measured the parameters $L(m)$ and $g(m)$ using our micro-benchmarks. We calculated RTT across both the EIB and the BIF by averaging the RTT for communications between SPE0 and all other SPEs. Recall that for 16 SPEs (Cell Blade), there are two gap parameters: one for EIB messages, g_1 and one for BIF messages, g_2. Figure 7 shows the gap parameter values for various message sizes. The g_1 value is different from g because there are additional SPEs across the BIF posting bus requests to the data arbiter. Therefore the waiting time for each SPE request is increased, Recall that we use g when only one Cell processor is involved in the communication.

The congestion parameter (C), is calculated by the butterfly communication pattern micro-benchmark, with each SPE sending and receiving data from another distinct SPE. Note that in Figure 8, there is almost no congestion up to a data size of 256 bytes. The latency starts to climb as the data size increases above 256 bytes and as the number of data messages crossing the BIF increases above two. The latency increase caused by congestion is up to 218% of the latency when there is no contention.

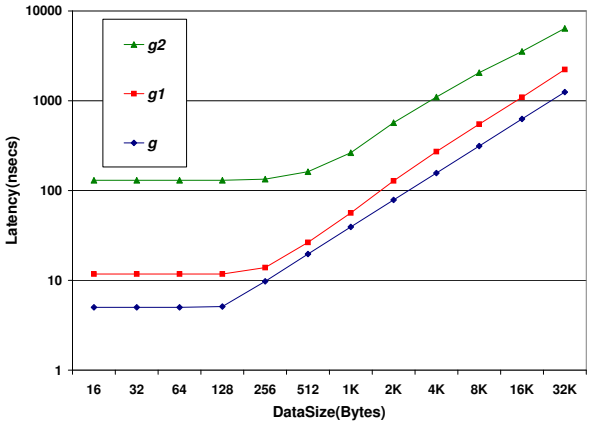

Figure 7: Gap micro-benchmark results.

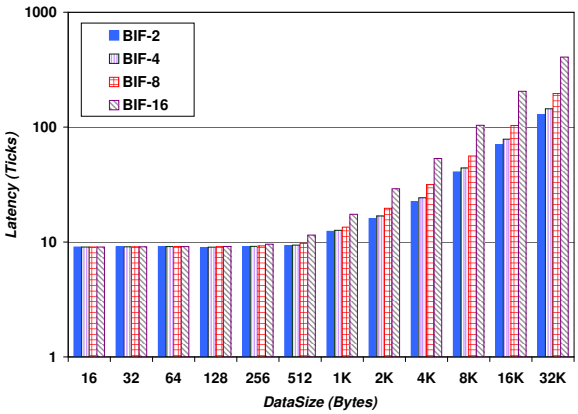

Figure 8: BIF Contention micro-benchmark results.

7.2 Analysis of Collective Communication performance models

All of the execution times predicted by the performance models are very close to the experimental results. Figure 9 shows the experimentally measured and predicted values of each barrier implementation. In the gather-broadcast barrier, the actual performance is slightly worse than the model because of some control code in the implementation; in the Bruck barrier, the actual performance is slightly better due to message pipelining effects not captured by the model. These errors are small enough that these predictions can be used to make correct decisions about which algorithm to choose.

Figure 10 shows the error rate for different algorithms for all-gather, broadcast, reduce and all-reduce. Considering all algorithms, implementations, and data sizes, we have a total of 425 data points. Of those, 398 (94%) have a discrepancy between the model and the actual execution of less than 10% and all of them see errors less than 15%. The algorithms not shown in this figure have error rates less than 5%. All of the error rate graphs show data sizes from 64 bytes to 32 KB. Performance of data sizes less than 64 bytes is the same as the performance at 64 bytes. Again, the errors are small enough that the predictions can still be used to make correct decisions when selecting algorithms. We also modeled and tested Barrier and Broadcast up to 64 SPEs (4 Cell Blades) to show that our models can be easily plugged into cluster-based collective communication models. The error rates were within 13%.

We attribute the general accuracy of the models to two factors: first, the accurate micro-benchmarks and methodology we developed to capture latency, gap and congestion parameters on the Cell; and second, the breaking down of each algorithm into fundamental components that could be represented using 3 basic and easily modeled communication patterns. Almost all the algorithm predictions are very accurate, with a correlation factor of over 0.98 across all algorithms and implementations.

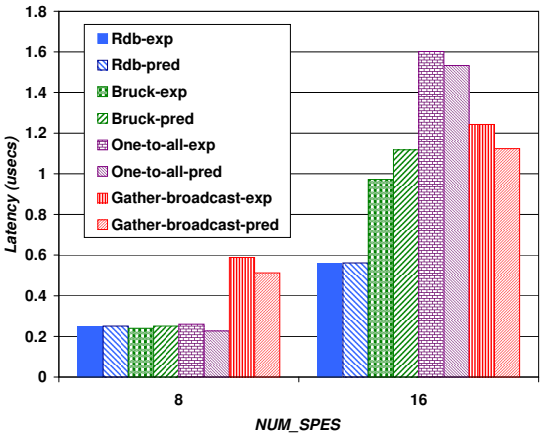

Figure 9: Performance predictions and actual latencies for various barrier implementations.

Figure 11 shows the error rates using non-Cell-specific modeling representative of the prior state-of-the-art. These models do not incorporate BIF contention, and they use non-Cell-specific micro-benchmarks for defining and measuring g, following the methods of Kielmann et al. [13]. For some of the algorithms and data sizes shown, errors are below 1%. However, in other cases, errors go as high as 140% as a result of using the old gap parameters or 56% from ignoring BIF contention. Because the errors are both large and variable, these models cannot be used to select appropriate algorithms before implementing the collective operations.

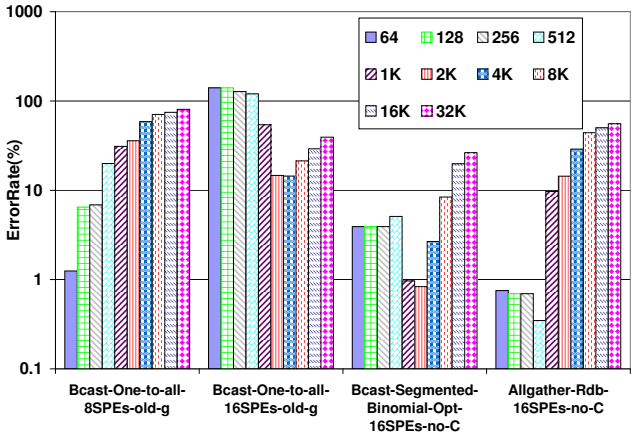

Figure 11: Error rates when using non-Cell-specific models.

8. Related work

Several approaches have been proposed in the literature to model collective communication, especially in the context of MPI collective communication.

Kielmann et al. have presented work on modeling hierarchical collective communication and discussed the limitations in existing performance models that motivated their new model called

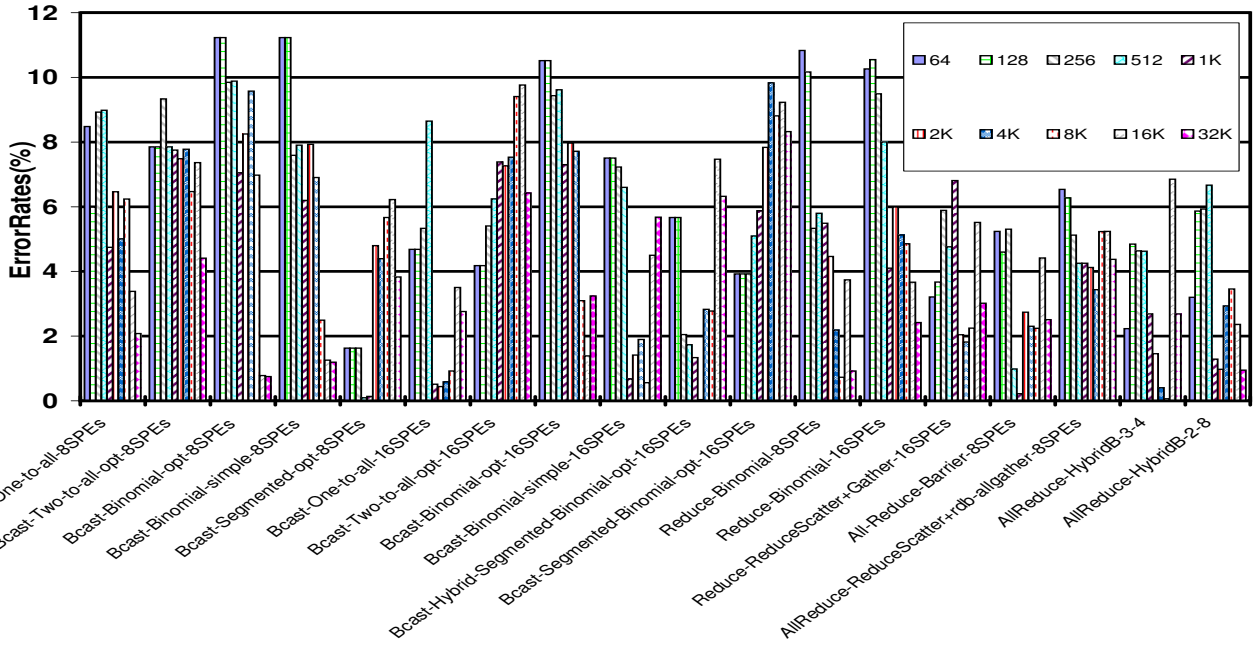

Figure 10: Predictions of various algorithms.

PLogP [13]. Our work is similar in that our models must account for the interconnect characteristics of the design and the ways in which algorithms use that interconnect; however, the Cell Blade is a unique design and thus requires a different approach than the modeling of clusters on a LAN or WAN.

Besides LogP and PLogP, several other performance models exist, such as LogGP and the Hockney model [3, 12]. Thakur et al. use the Hockney model to analyze the performance of various collective communication algorithms [20, 21]. Moritz et al. use the LoGPC model to model contention in message passing programs [15]. Unfortunately, its complexity makes it hard to apply such a model in practical situations. It also assumes k-ary n-cube networks which might not be true in some cases, such as the BIF in Cell Blades. Barchet et al. use PlogP to model all-to-all communication and extend it to include a congestion parameter [5] similar to what we do. Our methods to capture gap and contention are different from all of the above methods. Pjesivac-grbovic et al. have evaluated the Hockney, LogP, LogGP, and PLogP standard models for collective communication [17]. Their approach consists of modeling each collective communication using many different algorithms, and they give an optimal/optimized algorithm for different communication models. They all ignore contention, and this may lead to inaccurate results in environments, such as the Cell blade, that heavily stress the interconnect.

Faraj et al. propose an automatic generation and tuning of MPI collective communication routines using the Hockney model [11]. Their modeling accounts for network contention but also requires a large number of parameters, many of which are difficult to measure for some networks and processor architectures.

Despite the various related efforts, there is currently no communication model in the literature that targets collective communication on accelerator-based processors such as the Cell.

9. Conclusions

This paper presents performance models for some advanced collective communication algorithms that exploit features of the Cell architecture. In particular we provide an extension to the PLogP performance model to predict the performance of various collective communication operations. This modeling is achieved by breaking down the collective communication algorithms into three basic communication patterns that can be analyzed and modeled directly. Expressing and modeling collectives in this fashion enables the extension of this work to incorporate other algorithms that can also be built on top of those three primitives. The predictions from these performance models are accurate and the errors are within 10% for nearly all of the algorithms modeled, both within a single Cell chip and across Cell chips that are part of a blade. We show how these models can be extended for a system that consists of a cluster of Cell Blades. We also discuss how these models can be extended for Roadrunner-like systems.

The future of HPC depends on efficient communication algorithms and the models presented in this paper will serve as a guideline for future developers of any collective communication pattern for Cell-based systems. There exists a myriad of algorithms for any given collective communication operation, and selecting, tuning, and coding the best algorithm for a given platform requires substantial development time. Using the model equations to predict performance and communication efficiency allows developers to focus on the design of an algorithm rather than going through the tedious process of implementing the algorithm. Although this is true of modeling in general, it is even more so for modeling the Cell since Cell programming is often substantially different from, and more complicated than, general-purpose programming. Modeling can thus play an important role in bridging the gap between algorithm design and performance programming.

10. 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 would also like to thank Scott Pakin at LANL for explaining collective communication algorithms in CML and providing valuable feedback regarding Cell/Roadrunner.

References

[1] http://www.mcs.anl.gov/research/projects/mpich2.

[2] T. Ainsworth and T. Pinkston. On characterizing performance of the Cell broadband engine element interconnect bus. *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 18–29, May 2007.

[3] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. Loggp: Incorporating long messages into the logp model — one step closer towards a realistic model for parallel computation. Technical report, Santa Barbara, CA, USA, 1995.

[4] Q. Ali, S. P. Midkiff, and V. S. Pai. Efficient high performance collective communication for the cell blade. In *ICS '09: Proceedings of the 23rd International Conference on Supercomputing*, pages 193–203, New York, NY, USA, 2009. ACM.

[5] L. Barchet-Steffenel and G. Mounie. Total exchange performance modelling under network contention. In *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics*, LNCS Vol. 3911, pages 100–107, 2005.

[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] J. Bruck, C. tien Ho, S. Kipnis, E. Upfal, 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.

[8] 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.

[9] D. Culler, R. K. Y, D. Patterson, A. Sahay, R. Subramonian, and T. V. Eicken. LogP: Towards a realistic model of parallel computation. In *In Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, 1993.

[10] D. E. Culler, L. T. Liu, R. P. Martin, and C. O. Yoshikawa. Assessing fast network interfaces. *IEEE Micro*, 16(1):35–43, 1996.

[11] A. Faraj and X. Yuan. Automatic generation and tuning of MPI collective communication routines. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 393–402, New York, NY, USA, 2005. ACM.

[12] R. W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Computing*, 20(3):389–398, 1994.

[13] 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.

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

[15] C. A. Moritz and M. I. Frank. LoGPC: Modeling network contention in message-passing programs. *IEEE Transactions on Parallel and Distributed Systems*, 12(4):404–415, 2001.

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

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

[18] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing Journal*, 10:127–143, 2007.

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

[20] R. Thakur and W. Gropp. Improving the performance of collective operations in MPICH. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. Number 2840 in LNCS, Springer Verlag (2003) 257267 10th European PVM/MPI Users Group Meeting*, pages 257–267. Springer Verlag, 2003.

[21] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, February 2005.