Fast and Efficient Synchronization and Communication Collective Primitives for Dual Cell-Based Blades*

Epifanio Gaona, Juan Fernández, and Manuel E. Acacio

Dept. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, Spain
{fanios.gr,juanf,meacacio}@ditec.um.es

Abstract. The Cell Broadband Engine (Cell BE) is a heterogeneous multi-core processor specifically designed to exploit thread-level parallelism. Its memory model comprehends a common shared main memory and eight small private local memories. Programming of the Cell BE involves dealing with multiple threads and explicit data movement strategies through DMAs which make the task very challenging. This situation gets even worse when dual Cell-based blades are considered. In this context, fast and efficient collective primitives are indispensable to reduce complexity and optimize performance.

In this paper, we describe the design and implementation of three collective operations: barrier, broadcast and reduce. Their design takes into consideration the architectural peculiarities and asymmetries of dual Cell-based blades. Meanwhile, their implementation requires minimal resources, a signal register and a buffer. Experimental results show low latencies and high bandwidths, synchronization latency of 637 ns, broadcast bandwidth of 38.33 GB/s for 16 KB messages, and reduce latency of 1535 ns with 32 *floats*, on a dual Cell-based blade with 16 SPEs.

1 Introduction

The Cell BE was jointly developed by Sony, Toshiba and IBM to provide improved performance on game and multimedia applications [1]. However, there is a growing interest in using the Cell BE for high-performance computing due to its tremendous potential in terms of theoretical peak performance. From the architectural point of view, the Cell BE can be classified as a heterogeneous multi-core processor specifically designed to exploit thread-level parallelism. As for its memory model, the Cell BE comes with a hybrid scheme with a common shared main memory and eight small private local memories. The combination of all these factors makes programming of the Cell BE a really complex task.

* This work has been jointly supported by the Spanish MEC under grants "TIN2006-15516-C04-03" and European Comission FEDER funds under grant "Consolider Ingenio-2010 CSD2006-00046". Epifanio Gaona is supported by fellowship 09503/FPI/08 from Comunidad Autónoma de la Región de Murcia (Fundación Séneca, Agencia Regional de Ciencia y Tecnología).

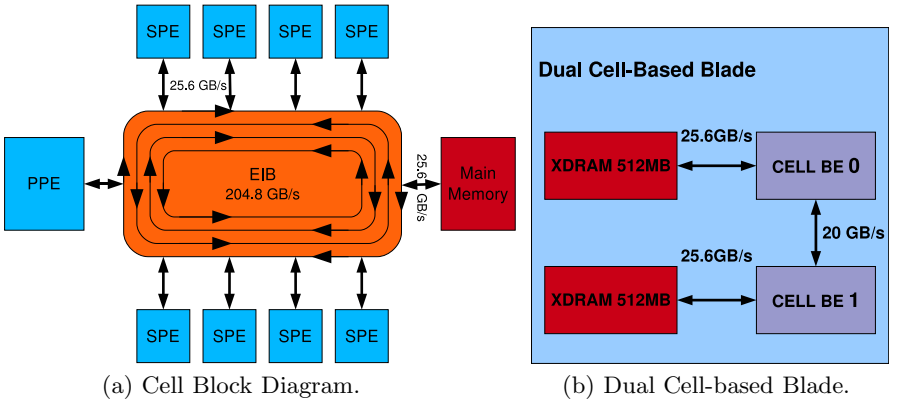
H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 900–911, 2009.

Cell BE programmers must explicitly cope with multiple threads that have to orchestrate frequent data movements to overlap computation and communication due to the small size of the private local memories [2].

In this scenario, a number of programming models and platforms have been proposed for the Cell BE. Some of them are based on well-known shared-memory and message-passing libraries such as OpenMP [3] and MPI [4,5]. In the meantime, others such as CellSs [6] and RapidMind [7] are based on task-parallel and stream programming models, respectively. Anyway, synchronization and coordination of multiple threads is a common source of programming errors and performance bottlenecks [8]. For that reason, these programming models either offer explicit synchronization and communication collective primitives, such as `MPI_Barrier` or `#pragma omp barrier`, in order to make code less error prone. In any case, fast and efficient collective primitives are clearly needed.

The Cell BE provides programmers with a broad variety of communication and synchronization primitives between the threads that comprise parallel applications, such as DMAs, mailboxes, signals and atomic operations, which were evaluated by us in [9] for dual Cell-based blades. Nevertheless, the Cell BE SDK provided by IBM does not provide programmers with synchronization and communication collective primitives similar to those available for many other parallel systems [10]. To the best of our knowledge, the work presented in [11] is the only one that tackles this problem. However, it focuses on the MPI programming model for the Cell BE introduced by [5] considering that application data is stored in main memory and is not optimized for dual Cell-based Blades. In contrast, our proposal in this work is more general and efficient. First, it assumes that data resides in the SPEs' LSs. Second, it has been specifically optimized for dual Cell-based blades taking advantage of the know-how acquired in [9] and, therefore, improves barrier performance results (see Section 4). Third, it requires less resources to implement the very same set of collective primitives.

Our main contributions are: (1) a description of the design of several algorithms for three common collective operations: barrier, broadcast and reduce;



(a) Cell Block Diagram. (b) Dual Cell-based Blade.

Fig. 1. Cell BE Architecture

(2) an optimized implementation of the algorithms for dual Cell-based blades requiring minimal resources, a signal register and a single 16 KB buffer; and, (3) a comparative analysis of the performance of the algorithms. Note that the use of a single 16 KB buffer limits the maximum broadcast and reduce sizes. This limitation is not such a restriction due to the small size of the LSs and the fact that reductions involving simple operations on small data sizes are the prevalent case in most scientific applications [12]. Finally, as an outcome of this work, Cell BE programmers should be able to implement the most appropriate version of these primitives depending of their expertise and performance requirements.

The rest of the paper is organized as follows. In Section 2, we provide a revision of the architecture and programming of dual Cell-based blades. Next, in Section 3 our proposal for designing and implementing the three collective primitives is explained. Then, the experimental results obtained on a dual Cell-based blade are presented in Section 4. Finally, Section 5 gives the main conclusions of this work.

2 Cell BE: Architecture and Programming

The Cell Broadband Engine (Cell BE) [1] is an heterogeneous multi-core chip composed of one general-purpose processor, called *PowerPC Processor Element* (PPE), eight specialized co-processors, called *Synergistic Processing Elements* (SPEs), a high-speed memory interface controller, and an I/O interface, all integrated in a single chip. All these elements communicate through an internal high-speed *Element Interconnect Bus* (EIB) (see Figure 1(a)).

A dual Cell-based blade is composed of two separate Cell BEs linked together through their EIBs as shown in Figure 1(b). The EIB is extended transparently across a high-speed coherent interface running at 20 GBytes/second in each direction [2]. In this configuration the two Cell BEs operate in SMP mode with full cache and memory coherency.

The Cell BE provides programmers with a variety of synchronization and communication mechanisms: *DMA Transfers*, *Atomic Operations*, *Mailboxes* and *Signals*.

SPEs use DMA transfers to read from (GET) or write to (PUT) main memory, or to copy data between the eight SPEs' LSs (MOV). DMA transfer size must be 1, 2, 4, 8 or a multiple of 16 Bytes up to a maximum of 16 KB. DMA transfers can be either blocking or non-blocking. The latter allow to overlap computation and communication: there might be up to 128 simultaneous transfers between the eight SPEs' LSs and main memory. In all cases, peak performance can be achieved when both the source and destination addresses are 128-Byte aligned and the size of the transfer is an even multiple of 128 Bytes [13].

Read-modify-write atomic operations enable simple transactions on single words residing in main memory. For example, the *atomic_add_return* atomic operation adds a 32-bit integer to a word in main memory and returns its value before the addition.

Mailboxes are FIFO queues that support exchange of 32-bit messages among the SPEs and the PPE. Each SPE includes two outbound mailboxes, called *SPU Write Outbound Mailbox* and *SPU Write Outbound Interrupt Mailbox*, to send messages from the SPE; and a 4-entry inbound mailbox, called *SPU Read Inbound Mailbox*, to receive messages.

In contrast, signals were designed with the only purpose of sending notifications to the SPEs. Each SPE has two 32-bit signal registers to collect incoming notifications, namely *SPU Signal Notification 1 (SIG1)* and *SPU Signal Notification 2 (SIG2)*. A signal register is assigned a MMIO register to enable remote SPEs and the PPE to send individual signals (*overwrite mode*) or combined signals (*OR mode*) to the owner SPE. It is worth noting that the latter mode allows to collect and identify incoming notifications from remote SPEs. To do so, it is enough to assign the i*th* bit of a signal register to the i*th* SPE, that is, SPE i can signal SPE j by sending the value 2^{id} which would set the i*th* bit of the target SPE j's signal register. In this way, SPE j can determine the number and identity of the remote SPEs by simply using bit masks. This strategy revealed extremely useful to implement collective primitives as we will see in Section 3.

Cell BE programming requires separate programs, written in C/C++, for the PPE and the SPEs, respectively. The PPE program can include extensions (e.g., `vec_add`), to use its VMX unit; and library function calls, to manage threads and perform communication and synchronization operations (e.g., `spe_create_thread` and `spe_write_in_mbox`). The SPE program follows an SPMD model. It includes extensions, to execute SIMD instructions, and communication and synchronization operations (e.g., `spu_add` and `mfc_get`).

Programming of a dual Cell-based blade is similar to that of an independent Cell from a functional point of view. However, there are two important differences. Firstly, dual Cell-based blades have 16 SPEs at programmer's disposal rather than 8 SPEs. This feature doubles the maximum theoretical peak performance but also makes much more difficult to fully exploit thread-level parallelism. Secondly, from an architectural point of view, synchronization and communication operations crossing the inter-Cell interface result in significantly less performance than those that stay on-chip [9]. This feature is a key factor that must be taken into consideration to avoid unexpected and undesirable surprises when designing synchronization and communication collective primitives for a dual Cell-based blade platform.

3 Design and Implementation

This section describes the design and implementation of several alternative algorithms for three common collective synchronization and communication primitives: barrier, broadcast and reduce. Most of them are based on a three-phase scheme. In the *ready* phase all group members wait for each other to reach the collective primitive call. The *transfer* phase is executed in between the other two phases and is devoted to transferring data among SPEs' LSs using DMAs when necessary. Finally, in the *go* phase all group members are informed to resume

computation. Unless otherwise noted, only one signal register in *OR mode* is needed to implement the *ready* and *go* phases as described in Section 2. Moreover, even though it is not discussed in the paper, the 16 most significant bits (*msb*) and the 16 less significant bits (*lsb*) of such a signal register have been used alternatively in order to ensure correctness between consecutive executions of each primitive when necessary. Note that the second signal register could be used for synchronizing up to 4 Cell chips. A single 16 KB buffer is in turn employed in the *transfer* phase. Finally, in all cases, there is a single group that comprehends all threads run by the SPEs used by the application.

3.1 Barrier

A barrier synchronization blocks the callers until all group members have invoked the primitive. We have implemented three different algorithms and two variants for the first two ones (see Figure 2).

All-To-One Barrier (ATOBar). In the *ready* phase every SPE signals the root SPE (see the arrows labeled with number 1 in Figure 2(a)). The root SPE in turn waits for all SPEs to enter the barrier and then clears its signal register. In the *go* phase, the root SPE signals all SPEs (see the arrows marked with number in 2 Figure 2(b)). As SPEs receive the acknowledgement from the root SPE, they resume computation.

All-To-One Barrier for Blades (ATOBarB). This algorithm introduces an optimization over the previous one in order to reduce the inter-Cell traffic when the second Cell comes into play. To do so, the ATOBar algorithm is executed locally by both Cells so that both root SPEs must signal each other (inter-Cell synchronization) before the beginning of the *go* phase. Consequently, only two signals are sent through the inter-Cell interface (see the arrows marked with number 2 in Figure Figure 2(b)).

Tree Topology Barrier (TTBar). This scheme is similar to the ATOBar algorithm but a tree structure is used in the upward and downward directions for the *ready* and *go* phases, respectively (see Figure Figure 2(c)). The algorithm requires $2 \times \lceil \log_d N \rceil$ steps where d is the degree of the tree.

Tree Topology Barrier for Blades (TTBarB). This version follows the same idea as the ATOBarB algorithm but using instead the TTBar algorithm for synchronization inside each Cell (as illustrated in Figure Figure 2(d)). In this case, the algorithm employs $2 \times \lceil \log_d \frac{N}{2} \rceil + 1$ steps.

Pairwise Exchange Barrier (PEBar). Parwise exchange is a well-known recursive algorithm [14]. Unlike the previous schemes, all SPEs are progressively paired up to synchronize in each round. Let N be the number of SPEs. At step s, SPE i and SPE j, where $j = i \oplus 2^s$, signal each other. If the number of SPEs is a power of two, then the algorithm requires $\log_2 N$ steps. Otherwise, PE needs $\lfloor \log_2 N \rfloor + 2$ steps [14]. Let M be the largest power of two less than N. First, $\forall k \geq M$, SPE k signals SPE i, where $i = k - M$. Second, $\forall i < M$,

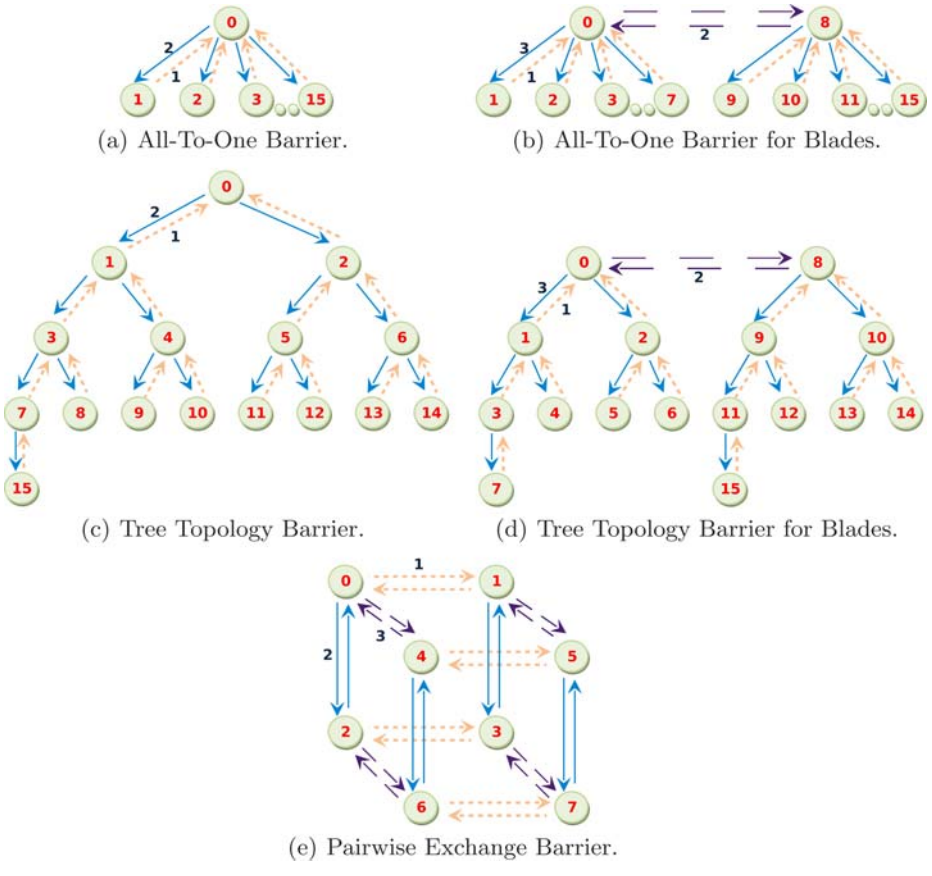


Fig. 2. Barrier Algorithms: *ready*, inter-Cell synchronization, and *go* phases

SPE i executes the PE algorithm. Third, $\forall k \geq M$, SPE i signals SPE k, where $i = k - M$, upon completion of the intermediate PE synchronization. Figure 2(e) shows these three steps.

3.2 Broadcast

A broadcast communication is similar to a barrier synchronization in which all group members must receive a message from the root member before they leave the barrier. The broadcast algorithms presented here are based on a three-phase scheme: *ready*, *broadcast* and *go*. The *ready* and *go* phases are similar to those of barrier algorithms. In the meantime, the *transfer* phase becomes a *broadcast* phase where a message is MOVed from the source SPE's LS to the destination SPEs' LSs. The broadcast buffers always reside at the same LS locations in all SPEs. Thus, their effective addresses can be computed by just adding the effective starting address of the corresponding SPEs' LSs. This simplification

is not a hard constraint since different effective destination addresses could be easily gathered in the *ready* phase. Broadcast message size is limited to 16 KB, that is, the maximum DMA size, and a 16 KB buffer per SPE is hence needed. Note that broadcast messages greater than 16 KB could be sent by using DMA lists or by implementing a flow control mechanism (e.g. through mailboxes) in charge of monitoring the transmission of the segments of the message.
We have implemented two different algorithms (see Figure 3), the broadcast versions of the ATOBar and TTBar algorithms, with a number of variants each through incorporating some optimizations. PE broadcast version is not considered because the *ready* and *go* phases cannot be decoupled. Description of non-blade, unoptimized versions of ATOBcastB$_{Opt}$ and TTBcastB$_{Opt}$ have been omitted since their implementations are straightforward from the description of the corresponding barrier algorithms.

All-To-One Broadcast for Blades with Optimizations (ATOBcastB$_{Opt}$). This algorithm is similar to the ATOBar version with two major optimizations. On the one hand, both root SPEs run the ATOBcast algorithm but the root SPE on the second Cell is treated as another remote SPE of the root SPE on the first Cell. In this way, a single broadcast message and two signals go across the inter-Cell interface. On the other hand, the *broadcast* phase overlaps with the *ready* and *go* phases. When the root SPE is signaled by a remote SPE, it immediately starts an asynchronous DMA to transfer the broadcast message from the local LS to the corresponding remote SPE's LS. After signals from all remote SPEs have been received and the subsequent DMAs have been initiated, the root SPE waits for each DMA to complete and immediately signals the corresponding remote SPE, in the same order signals were received in the *ready* phase. As a minor optimization of the last step, the root SPE on the second Cell is given a highest priority than the other remote SPEs.

Tree Topology Broadcast for Blades with Optimizations (TTBcastB$_{Opt}$). This scheme is derived from the TTBar algorithm and incorporates similar optimizations to the ones used in ATOBcastB$_{Opt}$.

3.3 Reduce

A reduce communication behaves like a broadcast communication in which messages flow in the reverse order. The broadcast algorithms presented here are based on a three-phase scheme: *ready*, *gather* and *go*. The *ready* and *go* phases are identical to those of broadcast algorithms. In the *gather* phase incoming reduce messages, consisting of arrays of integers or single-precision floating-point numbers, are combined using a specific operator (e.g. ADD or SUB) through SIMD *intrinsics* (e.g. `spu_add` or `spu_sub`). Broadcast buffers are also used here for reduce operations. As in the case of broadcast, we have implemented two different algorithms (graphical representation would be equal to that of Figure 3 if arrows labeled with 2 appeared reversed), the reduce versions of the ATOBarB and TTBarB algorithms, with a number of variants each through incorporating the same optimizations used by the ATOBcastB$_{Opt}$ and TTBcastB$_{Opt}$ algorithms.

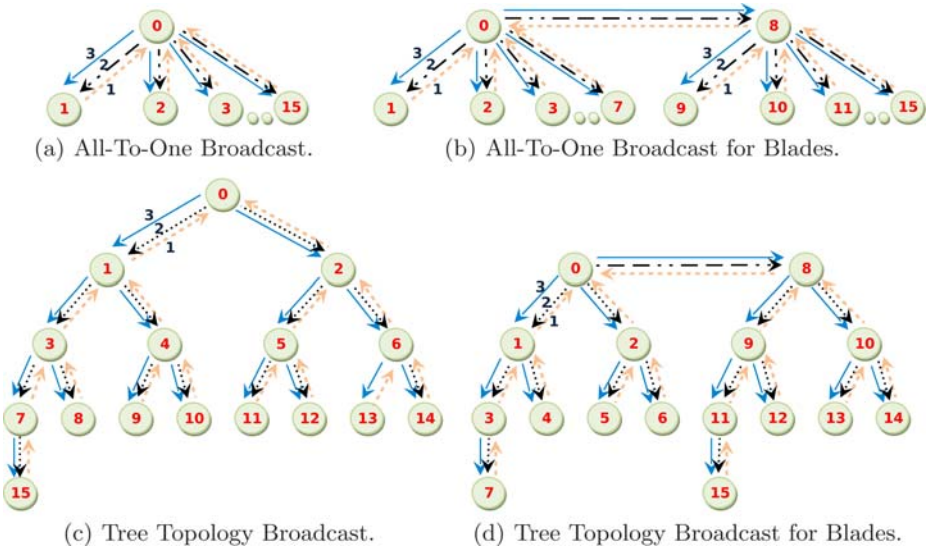


Fig. 3. Broadcast Algorithms: *ready*, *broadcast*, and *go* phases

Again, non-blade, unoptimized versions of the ATORedB$_{Opt}$ and TTRedB$_{Opt}$ algorithms have been left out since their implementations are straightforward from the description of the corresponding barrier algorithms.

All-To-One Reduce for Blades with Optimizations (ATORedB$_{Opt}$). This algorithm applies similar optimizations to the ones implemented by the ATOBcastB$_{Opt}$ algorithm to ATORed. First, both root SPEs run the ATORed algorithm but the root SPE on the second Cell becomes another remote SPE of the root SPE on the first Cell. Thus, it is guaranteed that no more than a single reduce message and two signals cross the inter-Cell interface. Second, the *gather* phase overlaps with the *ready* and *go* phases. Upon signal reception, the root SPE initiates an asynchronous DMA to transfer the reduce message from the remote SPE's LS into its local LS. When all signals from children SPEs have been processed, the root SPE proceeds in the same order they were received: (1) waits for DMA completion, (2) computes the reduce using local data and remote data, and (3) signals the corresponding remote SPE. Also, the root SPE on the second Cell is given a highest priority as a minor optimization. Finally, reduce operations are limited to 512 integers or single-precision floating point numbers and, therefore, reduce messages from all remote SPEs can be accommodated in the same buffer used by broadcast operations.

Tree Topology Reduce for Blades with Optimizations (TTRedB$_{Opt}$). This scheme is derived from the TTRed algorithm and incorporates similar optimizations to the ones used in ATOBcastB$_{Opt}$. Consequently, regardless of the degree of the tree, the root SPE on the second Cell is treated as a child of the root SPE on the first Cell, and the *gather* phase overlaps with the *ready* and *go*

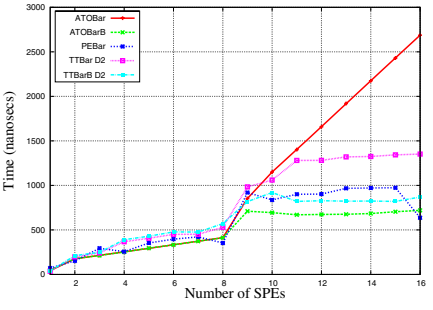


Fig. 5. Barrier Latency

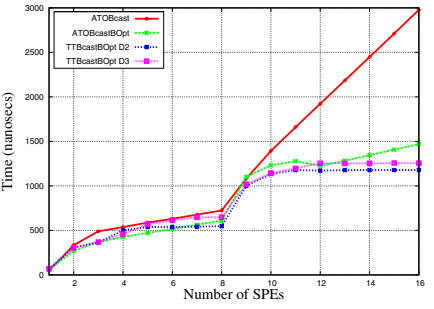


Fig. 6. Broadcast Latency (128 Bytes)

phases in the root and intermediate SPEs. In this case, reduce messages gathered from children SPEs are limited to 2,048 integers or single-precision floating point numbers in order not to exceed the maximum broadcast buffer size.

4 Performance Evaluation

This section details the making of the experiments and analyzes the experimental results obtained for the three collective primitives. In all experiments, the number of demanded SPEs invoke the specified collective primitive in a loop for one million iterations. Reported results are the best of three experiments executed in a row.

All the algorithms considered in this work have been implemented and evaluated using the IBM SDK v3.1 installed atop Fedora Core 9 on a dual Cell-based IBM BladeCenter QS20 blade which incorporates two 3.2 GHz Cell BEs v5.1, namely Cell0 and Cell1, with 1 GByte of main memory and a 670 GB hard disk.

4.1 Barrier

First of all, we present in Figure 5 the latency (measured in nanoseconds) for the implementations of the barrier primitive discussed in Subsection 3.1. As we can see, the naive All-To-One Barrier (ATOBar) is the best option when the number of SPEs involved is small (less than 8). In this case, all SPEs are in the same Cell chip and two signals, one from each SPE to the root SPE and another one in the opposite direction, suffice to complete the barrier primitive. Unfortunately, as the number of SPEs grows, so does the time taken by the ATOBar implementation, which becomes impractical when the number of SPEs is greater than 8. In this case, SPEs belonging to the two Cell chips participate in the barrier and latency is dominated by inter-chip signals (round-trip latencies of inter-chip and intra-chip signals are 619.4 ns and 160.1 ns [9], respectively). In order to cope with this problem, we proposed the All-To-One Barrier for Blades (ATOBarB) implementation. Up to 8 SPEs, the behavior of ATOBar and ATOBarB is exactly the same. From 9 SPEs up, two SPEs (one per Cell chip) act as roots and just two signals (exchanged by the two root SPEs once the local *ready* phases have been completed) must traverse the inter-Cell interface.

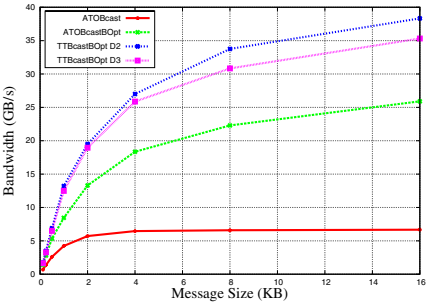


Fig. 7. Broadcast Bandwidth

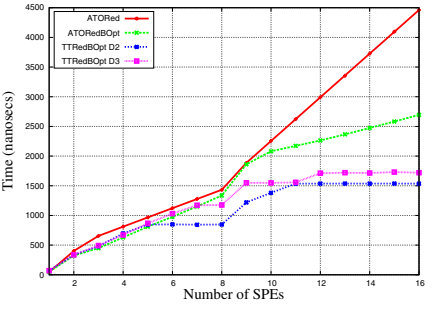


Fig. 8. Reduce Latency (32 floats)

As seen in Figure 5, this optimization ensures perfect scalability for more than 9 SPEs, since the two local *ready* and *go* phases are performed in parallel.

Similarly to ATOBar and ATOBarB, the Tree Topology Barrier (TTBar) and Tree Topology Barrier for Blades (TTBarB) implementations perform identically when the number of SPEs is lower than 9. Latency of TTBar and TTBarB increases afterwards due to the greater number of steps that are required to complete the *ready* and *go* phases. Observe, however, that they still scale much better than ATOBar. When more than 8 SPEs are involved, TTBarB avoids the scalability problems suffered by TTBar because the tree structure it implements for the *ready* and *go* phases reduces the number of inter-Cell signals, which finally translates into lower latency. Obviously, as the degree of the tree grows, performance of TTBarB progressively approaches to that of ATOBarB. For the sake of brevity, we present results just for degrees 2 and 3 though.

Finally, the Pairwise Exchange Barrier (PEBar) obtains the shortest latencies when the number of SPEs is a power of two because the number of steps is smaller than the tree topology algorithms. The extra steps required in the rest of the cases blur the advantages brought by the distributed way of synchronizing the SPEs that this algorithm entails. Note that all inter-chip signals are sent during the same phase with this algorithm which explains the almost constant latency increase when the number of SPEs is greater than 8.

4.2 Broadcast

Latency of the implementations of the broadcast primitive discussed in Subsection 3.2 is plotted in Figure 6 for 128-byte packets with 16 SPEs. Latency for larger packet sizes up to 16 KB takes the very same shape, although the absolute differences in terms of latency become more acute. Consequently, those results have been omitted for the sake of brevity.

As it can be seen in Figure 6, TTBcastBOpt obtains the best results in terms of latency when the number of SPEs is greater than 6, especially with a degree 2 tree. The tree structure assumed in TTBcastBOpt allows to initiate several DMA transfers simultaneously thus reducing the time employed in the *broadcast* phase. As expected, the larger the packet size, the lower the number of SPEs for

which TTBcastBOpt performs better than ATOBcast. For example, for 16-KB packets TTBcastBOpt is the best scheme when 3 or more SPEs are involved.

Additionally, Figure 7 shows the effective bandwidth (measured in GB/s) that is reached for the same implementations of the broadcast primitive as packet size varies from 128 bytes to 16 KB with 16 SPEs. As expected, absolute differences in terms of effective bandwidth grow with packet size. For example, a maximum bandwidth of 38.33 GB/s is reached with TTBcastBopt D2, whereas just 25 GB/s can be extracted from ATOBcastBopt.

4.3 Reduce

The latency of the reduce operations is shown in Figure 8. In all cases, results for reductions of 32-float vectors are presented. As already commented on, we strongly believe that this size is representative of most scientific applications, for which reductions involving simple operations on small data sizes are the prevalent case [12].

As it happens with barrier and broadcast primitives, the implementation of the reduce primitive with lowest latency depends on the total number of intervening SPEs. When the number of SPEs is less than 6, the simple ATORedB$_{Opt}$ scheme proves to be the best option. However, as the number of SPEs increases TTRedB$_{Opt}$ obtains lower latencies. When all SPEs are involved in the reduction, latency of ATORedB$_{Opt}$ almost doubles the figures obtained with TTRedB$_{Opt}$. The fact that partial reductions distributed among all SPEs (instead of being centralized in a couple of root SPEs) are carried out in parallel is the key factor to explain such a latency decrease.

Apart from degree 2 trees, we have also evaluated a version of TTRedB$_{Opt}$ with a degree 3 tree. However, as it can be observed in Figure 8, no latency advantage is found for TTRedB$_{Opt}$ D3, which performs slightly worse than TTRedB$_{Opt}$ D2 in all cases.

5 Conclusions

In this paper, we have described the design and implementation of three common collective operations: barrier, broadcast and reduce. For each of them we have explored several alternatives ranging from the naive approach to well-known algorithms coming from the cluster arena. Besides, we have adapted those in order to consider the architectural peculiarities and asymmetries of dual Cell-based blades. Our designs are efficient because, as we have explained, only require a signal register and a single 16 KB buffer. At the same time, our implementations of the algorithms are quite fast resulting in low latencies and high bandwidths for 16 SPEs. In addition, we have compared the performance of the different versions of each algorithm. This comparison highlights some interesting conclusions. On the one hand, the best algorithm is not always the same but depends on the number of SPEs and data size. On the other hand, the naive implementation obtains performance results close to the best algorithm in some cases.

In this way, experienced Cell BE programmers could opt for hybrid implementations that use the best algorithm depending on the input parameters. In the meantime, non-experienced programmers could adopt algorithms representing a tradeoff between performance and programming complexity.

References

1. Kahle, J., Day, M., Hofstee, H., Johns, C., Maeurer, T., Shippy, D.: Introduction to the Cell Multiprocessor. IBM Journal of Research and Development 49(4/5), 589–604 (2005)
2. Nanda, A., Moulic, J., Hanson, R., Goldrian, G., Day, M.N., D'Amora, B.D., Kesavarapu, S.: Cell/B.E. blades: Building blocks for Scalable, real-time, interactive, and digital media servers. IBM Systems Journal 51(5), 573–582 (2007)
3. O'Brien, K., O'Brien, K., Sura, Z., Chen, T., Zhang, T.: Supporting OpenMP on Cell. International Journal of Parallel Programming 36(3), 287–360 (2008)
4. Ohara, M., Inoue, H., Sohda, Y., Komatsu, H., Nakatani, T.: MPI microtask for programming the Cell Broadband Engine^{TM} processors. IBM Systems Journal 45(1), 85–102 (2006)
5. Kumar, A., Senthilkumar, G., Krishna, M., Jayam, N., Baruah, P.K., Sharma, R., Srinivasan, A., Kapoor, S.: A Buffered-mode MPI Implementation for the Cell BE^{TM} Processor. In: 7th International Conference on Computational Science, Beijing, China (2007)
6. Bellens, P., Prez, J.M., Bada, R.M., Labarta, J.: CellSs: a Programming Model for the Cell BE Architecture. In: Proceedings of IEEE/ACM Conference on SuperComputing, Tampa, FL (2006)
7. McCool, M.D.: Data-Parallel Programming on the Cell BE and the GPU using the RapidMind Development Platform. In: Proceedings of GSPx Multicore Applications Conference, Santa Clara, CA (2006)
8. McCool, M.D.: Scalable Programming Models for Massively Multicore Processors. Proceedings of the IEEE 96(5), 816–831 (2008)
9. Abellán, J.L., Fernández, J., Acacio, M.E.: Characterizing the Basic Synchronization and Communication Operations in Dual Cell-Based Blades. In: 8th International Conference on Computational Science, Krákow, Poland (2008)
10. Yu, W., Buntinas, D., Graham, R.L., Panda, D.K.: Efficient and Scalable Barrier over Quadrics and Myrinet with a New NIC-based Collective Message Passing Protocol. In: Proceedings of Workshop on Communication Architecture for Clusters, Santa Fe, NM, USA (2004)
11. Velamati, M.K., Kumar, A., Jayam, N., Senthilkumar, G., Baruah, P., Sharma, R., Kapoor, S., Srinivasan, A.: Optimization of Collective Communication in Intra-Cell MPI. In: Proceedings of 14th International Conference on High Performance Computing, Goa, India (2007)
12. Petrini, F., Moody, A., Fernández, J., Frachtenberg, E., Panda, D.K.: NIC-based Reduction Algorithms for Large-Scale Clusters. International Journal of High Performance Computing and Networking 4(3–4), 122–136 (2005)
13. Kistler, M., Perrone, M., Petrini, F.: Cell Processor Interconnection Network: Built for Speed. IEEE Micro. 25(3), 2–15 (2006)
14. Hoefler, T., Mehlan, T., Mietke, F., Rehm, W.: A Survey of Barrier Algorithms for Coarse Grained Supercomputers. Technical report, Technical University of Chemnitz (2004)