

## Scheduling Asymmetric Parallelism on a PlayStation3 Cluster

Filip Blagojevic<sup>1</sup>, Matthew Curtis-Maury<sup>1</sup>, Jae-Seung Yeom<sup>1</sup>,  
Scott Schneider<sup>1</sup> and Dimitrios S. Nikolopoulos<sup>1,2</sup>

<sup>1</sup>Department of Computer Science  
Virginia Tech  
Blacksburg, VA 24061

{filip,mfcurt,jyeom,scschnei,dsn}@cs.vt.edu

<sup>2</sup>Institute of Computer Science – FORTH  
P.O. Box 1385, Heraklion  
GR – 71110, Greece

dsn@ics.forth.gr

### Abstract

*Understanding the potential and implications of asymmetric multi-core processors for cluster computing is necessary, as these processors are rapidly becoming mainstream components in HPC environments. In this paper we evaluate a Linux cluster of Sony PlayStation3 consoles, using microbenchmarks and bioinformatics applications. We proceed to develop a model and scheduling techniques for effective execution of parallel applications on this low-cost, yet unconventional HPC platform based on the Cell/BE processor. We present an analytical formulation of layered parallelism for clusters of asymmetric multi-core multi-processors and propose new co-scheduling heuristics for effectively executing MPI code with nested task and data parallelism on these systems. Our model has low execution time prediction error and is reliable in predicting optimal mappings of nested parallelism in MPI programs on the PS3 cluster. The presented co-scheduling heuristics reduce slack time on the accelerator cores of the PS3 and improve the performance of MPI applications by 1.7–2.7×, when compared against the native OS scheduler.*

### 1. Introduction

Cluster computing is already feeling the impact of multi-core processors [7]. Several highly ranked entries of the latest Top-500 list include clusters of commodity dual-core processors<sup>1</sup>. The availability of abundant chip-level and board-level parallelism changes fundamental assumptions that developers currently make while writing software for HPC clusters. While recent work has improved our understanding of the implications of small-scale symmetric multi-core processors on cluster computing [1], emerging

<sup>1</sup><http://www.top500.org>

asymmetric multi-core processors such as the Cell/BE and boards with conventional processors and hardware accelerators –such as GPUs–, are rapidly making their way into HPC clusters [13].

Understanding the implications of asymmetric multi-core processors on cluster computing and providing models and software support to ease the migration of parallel programs to these platforms is a challenging and relevant problem. This paper makes four contributions:

i) We conduct a performance analysis of a Linux cluster of Sony PlayStation3 (PS3) nodes. Our analysis reveals the sensitivity of computation and communication to the mapping of asymmetric parallelism to the cluster and the importance of coordinated scheduling across multiple layers of parallelism. Optimal scheduling of MPI codes on the PS3 cluster requires coordinated scheduling and mapping of at least three layers of parallelism (two layers within each Cell processor and an additional layer across Cell processors), whereas the optimal mapping and schedule changes with the application, the input data set, and the number of nodes used for execution.

ii) We present a comprehensive model of layered parallelism for clusters with asymmetric multi-core processors and proceed to adapt and validate this model on the PS3 cluster. The terms in the model capture heterogeneity in the computation and communication substrates, the impact of techniques for communication/computation overlap, and the impact of techniques for oversubscribing host cores to increase the degree of parallelism exposed to accelerators. Our evaluation of the model shows that it estimates execution time with an average error rate of 5.2%. The model pin-points optimal mappings of MPI applications to the PS3 cluster with remarkable accuracy.

iii) We propose and analyze user-level scheduling heuristics for co-scheduling threads running on the host cores (PPE) with threads running on the accelerator cores (SPE) of the

Cell/BE, to minimize slack on the accelerator cores. We show that co-scheduling algorithms yield significant performance improvements (1.7–2.7×) over the native OS scheduler in MPI applications. We also explore the trade-off between different co-scheduling policies that selectively spin or yield the host cores, based on runtime prediction of task execution lengths on the accelerator cores.

iv) We present a comparison between our PS3 cluster and an IBM QS20 blade cluster (based on Cell/BE), illustrating that despite important limitations in computational ability and the communication substrate, the PS3 cluster is a viable platform for HPC research and development.

The rest of this paper is organized as follows: Section 2 presents our experimental platform. Section 3 presents our performance analysis of the PS3 cluster. Section 4 presents our model of hybrid parallelism and its validation. Section 5 presents co-scheduling policies for clusters of asymmetric multi-core processors and evaluates these policies. Section 6 compares the PS3 cluster against an IBM QS20 Cell-blade cluster. Section 6 discusses related work. Section 7 concludes the paper.

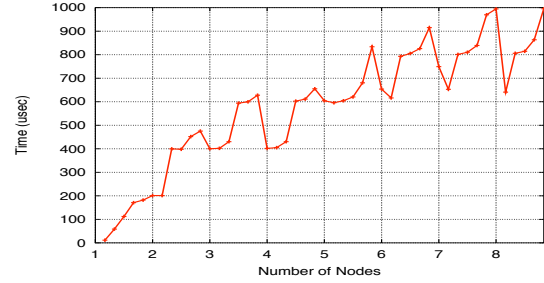
## 2 Experimental Platform

Our experimental platform is a cluster of 22 PS3 nodes. 8 of these nodes were available to us in dedicated mode for the purposes of this paper. The PS3 nodes are connected to a 1000BASE-T Gigabit Ethernet switch, which supports 96 Gbps switching capacity. Each PS3 runs Linux FC5 with kernel version 2.6.16, compiled for the 64-bit PowerPC architecture with platform-specific kernel patches for managing the heterogeneous cores of the Cell/BE. The nodes communicate with LAM/MPI 7.1.1. We used the IBM Cell SDK 2.1 for intra-Cell/BE parallelization of the MPI codes. Due to the PS3 design, the Linux kernel is required to run on top of a proprietary hypervisor. Though some devices are directly accessed, the built-in Gigabit Ethernet controller in the PS3 is accessed via hypervisor calls, therefore communication performance is not optimal.

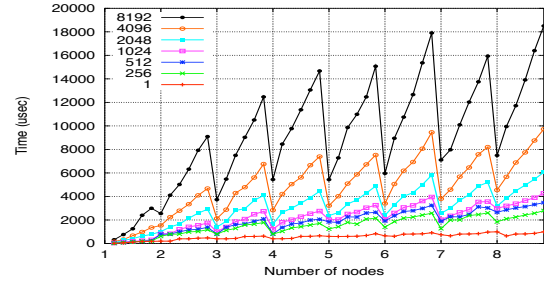
## 3 PS3 Cluster Scalability Study

### 3.1. MPI Communication Performance

To measure communication performance on the PS3 cluster, we use `mpptest` [11]. Due to space limitations, we present `mpptest` results only for two MPI communication primitives, which dominate communication time in our application benchmarks. Figure 1 shows the overhead of `MPI_Allreduce()`, with various message sizes. Each data point represents a number and a distribution of MPI processes between PS3 nodes. For any given number



(a) `MPI_Allreduce()` latency, one double.



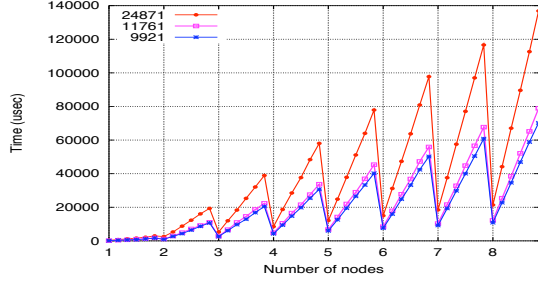
(b) `MPI_Allreduce()` latency, arrays of doubles.

**Figure 1.** `MPI_Allreduce()` performance on the PS3 cluster. Processes are distributed evenly between nodes. Each node runs up to 6 processes, using shared memory for communication within the node.

of PS3 nodes, we use 1 to 6 MPI processes, using shared memory for communication within the PS3. Our evaluation methodology stresses the impact of contention for communication bandwidth both within and across PS3 nodes. There is benefit in exploiting shared memory for communication between MPI processes on each PS3. For example, collective operations between 8 processes running on 2 PS3 nodes are 30% or more faster than collective operations between 8 processes running across 8 PS3 or 4 PS3 nodes. However, there is also a noticeable penalty for over-subscribing each PPE with more than two processes, due to OS overhead, despite our use of blocking shared memory communication within each PS3. Similar observations can be made for point-to-point communication (Figure 2).

### 3.2. Application Benchmarks

We evaluate the performance of two state-of-the-art computational phylogenetic tree inference codes, RAXML and PBPI, on our PS3 cluster. The applications have been painstakingly optimized for the Cell/BE [6]. Both RAXML and PBPI are capable of exploiting multiple levels of PPE and SPE parallelism. We used a task off-loading execution model in the codes. The execution commences on the PPE, and SPEs are used for accelerating computation-intensive loops. The off-loaded loops are parallelized across SPEs



**Figure 2.** `MPI_Send/Recv()` latency on the PS3 cluster. Processes are distributed evenly between nodes. Each node runs up to 6 processes, using shared memory for communication within the node.

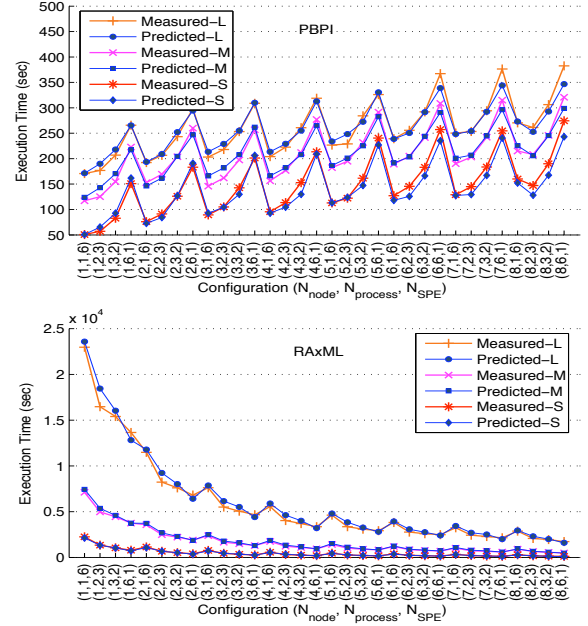
and vectorized within SPEs. The number of PPE processes and the number of SPEs per PPE process are user-specified.

When the PPE is oversubscribed with more than two processes, the processes are scheduled using an event-driven task-level parallelism (EDTLP) scheduler [5]. Each process executes until the point it off-loads an SPE task and then releases the PPE, while waiting for the off-loaded task to complete. The same process resumes execution on the PPE only after all other processes off-load SPE tasks at least once. The RAXML and PBPI ports on the PS3 cluster are adaptations of the original MPI codes and are capable for execution in a distributed environment.

For each application we use three data sets, briefly referred to as *small*, *medium*, and *large*. The large data set occupies the entire memory of a PS3, minus memory used by the operating system and the hypervisor. The medium and small data sets occupy 40% and 15% of the free memory of the PS3 respectively. In PBPI the small, medium, and large data sets represent 218 species with 1250, 3000, and 5000 nucleotides respectively. In RAXML, the small, medium, and large datasets represent 42, 50, and 107 species respectively. We execute PBPI using weak scaling, i.e. we scale the data set as we add more PS3 nodes, which is the recommended execution mode. For RAXML we use strong scaling, since the application uses a master-worker paradigm, where each worker performs independent, parameterized phylogenetic tree bootstrapping on the same input and processes the entire phylogenetic independently. Workers are distributed between nodes to maximize throughput. We perform 192 bootstraps, which is a realistic workload for real-world phylogenetic analysis.

Figure 3 illustrates the measured execution times of RAXML and PBPI on the PS3 cluster. The predicted execution times on the same charts are derived from our model, which is discussed in Section 4. We make three observations regarding measured performance:

i) The PS3 cluster scales well under strong scaling (RAXML) however it exhibits limited scalability under



**Figure 3.** Measured and predicted performance of applications on the PS3 cluster.  $x$ -axis notation:  $N_{node}$  - number of nodes,  $N_{process}$  - number of processes per node,  $N_{SPE}$  - number of SPEs per process.

weak scaling (PBPI), for the problem sizes considered. PBPI is more communication-bound than RAXML, as it involves several collective operations between executions of its Markov-chain Monte Carlo kernel. We note that due to the hypervisor of the PS3 and the lack of Cell/BE-specific optimization of the MPI library we used, the performance measurements on the PS3 cluster are conservative.

ii) The optimal layered decomposition of the applications is at the opposite ends of the optimization space. RAXML executes optimally if the PPE on each PS3 is oversubscribed by 6 MPI processes, each off-loading simultaneously on 1 SPE. PBPI generally executes optimally with 1 MPI process per PS3 using all 6 SPEs for data-parallel computation, as this configuration reduces inter-PS3 communication volume and avoids PPE contention.

iii) Although the optimal layered decomposition does not change with the problem size for the three data sets used in this experiment, it changes with the scale of the cluster. When PBPI is executed with 8 PS3s, the optimal operating point of the code shifts from 1 to 2 MPI processes per node, each off-loading simultaneously on 3 SPEs. We have verified with an out-of-band experiment that this shift is permanent beyond 8 PS3s. This shift happens because of the large drop in the per process overhead of `MPI_Allreduce()` (Figure 1), when 2 MPI processes are packed per node, on 3 or more PS3 nodes. This drop is large enough to outweigh the overhead due to contention between MPI processes on

the PPE. The difficulty in experimentally discovering the hardware and software implications on the optimal mapping of applications to clusters of asymmetric multi-core processors motivates the introduction of an analytical model.

## 4 Modeling Hybrid Parallelism

We present an analytical model of layered parallelism on clusters of asymmetric multi-core nodes, which is a generalization of a model of parallelism on stand-alone asymmetric multi-core processors which we proposed earlier [4]. Our generalized model captures computation and communication across nodes with *host cores* and *accelerator cores*. We specialize the model for the PS3 cluster, to capture the overhead of non-overlapped DMA operations, wait times during communication operations in the presence of contention for bandwidth both within and across nodes, and non-overlapped scheduling overhead on the PPEs.

We model the non-overlapped components of execution time on the Cell/BE's PPE and SPE, for single-threaded PPE code which off-loads to one SPE as:

$$T = (T_{ppe} + O_{ppe}) + (T_{spe} + O_{spe}) \quad (1)$$

where  $T_{ppe}$  and  $T_{spe}$  represent non-overlapped computation, while  $O_{ppe}$  and  $O_{spe}$  represent non-overlapped overhead on the PPE and SPE respectively. We apply this model for phases of parallel computation individually. Phases are separated by collective communication operations.

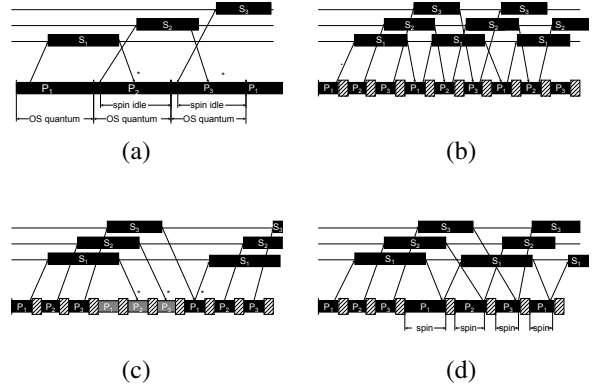
### 4.1. Modeling PPE Execution Time

The overhead on the PPE includes instructions and DMA operations to off-load data and code to SPEs, and wait time for receiving synchronization signals.

Assuming that multiple PPE threads can simultaneously off-load computation, we introduce an additional factor for context switching overhead on the PPE. This factor depends on the thread scheduling algorithm on the PPE. In the general case,  $O_{ppe}$  for code off-loaded from a single PPE thread to  $l$  SPEs is modeled as:

$$O_{ppe} = l \cdot O_{off-load} + T_{csw}(p) \quad (2)$$

We assume that a single PPE thread off-loads to multiple SPEs sequentially and that the context switching overhead is a function of the number of threads co-executed on the PPE, which is denoted by  $p$ .  $O_{off-load}$  is application-dependent and includes DMA setup overhead which we measure with microbenchmarks.  $T_{csw}$  depends on system software and includes context switching overhead for  $p/C$  context switches,  $C$  the number of hardware contexts on the PPE. The overhead per context switch is also measured with microbenchmarks.



**Figure 4.** Four cases illustrating the importance of co-scheduling PPE threads and SPE threads. PPE (P) threads poll shared memory locations directly to detect if a previously off-loaded SPE (S) thread has completed. Striped intervals indicate yielding of the PPE, dark intervals indicate computation leading to a thread off-load on an SPE, light intervals indicate computation yielding the PPE without off-loading. Stars mark cases of mis-scheduling.

If a hardware thread on the PPE is oversubscribed with multiple application threads, the computation time of each thread may increase due to on-chip resource contention. To accurately model this contention, we introduce a scaling parameter,  $\alpha(p)$  to the PPE computation component, which depends on the number of threads co-executed on the PPE. The PPE component of the model therefore becomes  $\alpha_p \cdot T_{ppe} + O_{ppe}$ . The factor  $\alpha_p$  is estimated using linear regression with one free parameter, the number of threads sharing a PPE hardware thread.

The formulation of  $T_{ppe}$  derived thus far ignores additional wait time of threads on the PPE due to lack of co-scheduling between a PPE thread and an SPE thread off-loaded from it. This scenario arises when the PPE hardware threads are time-shared between application threads, as shown in Figure 4(a), and is explained in detail in Section 5. Ideal co-scheduling requires accurate knowledge of the execution time of tasks on SPEs by both the operating system and the runtime system. This knowledge is not generally available. Our model assumes an idealized co-scheduling scenario. SPE tasks for a given phase of computation are assumed to be of the same execution length and are off-loaded in bundles with as many tasks per bundle as the number of SPEs on a Cell/BE. We also assume that the SPE execution time of the first task is long enough to allow for idealized co-scheduling, i.e. each PPE thread that off-loads a task is rescheduled on the PPE timely, to receive immediately the signal from the corresponding finishing SPE task. We explore this scheduling problem in Section 5 under more realistic assumptions and propose solutions.

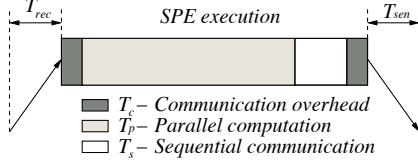


Figure 5. SPE execution

## 4.2. Modeling the off-loaded Computation

Execution on SPEs is divided into stages, as shown in Figure 5.  $T_{spe}$  is modeled as:

$$T_{spe} = T_p + T_s \quad (3)$$

$T_p$  denotes the computation executed in parallel by more than one SPE.  $T_s$  denotes the part of the off-loaded computation that is inherently sequential and cannot be parallelized. When  $l$  SPEs are used for parallelization of off-loaded code, the  $T_{spe}$  term becomes:

$$T_{spe} = \frac{T_p}{l} + T_s \quad (4)$$

The accelerated execution on SPEs includes three more stages, shown in Figure 5.  $T_{rec}$  and  $T_{sen}$  account for PPE-SPE communication latency, while  $T_c$  captures the SPE overhead that occurs when an SPE sends to or receives a message from the PPE. The per-byte latencies for  $T_{rec}$ ,  $T_{sen}$  and  $T_c$  are application-independent and are obtained from microbenchmarks.  $T_p$  and  $T_s$  are application-dependent and are obtained from a profile of a sequential run.

## 4.3. DMA Modeling

Each SPE on the Cell/BE is capable of moving data between off-chip memory and local storage as well as to and from the local storages of other SPEs, while executing computation. To overlap computation and communication, applications use double buffering.

When double-buffering is used, the off-loaded loop can be either computation or communication bound: if the amount of computation in a single iteration of the loop is sufficient to completely mask the latency of fetching the data necessary for the next iteration, the loop is computation bound. Otherwise the loop is communication bound.

Note that a parallel off-loaded loop can be described using Equation 4, independently of whether the parallel part of the loop is computation or communication bound. In both cases, the loop iterations are assumed to be distributed evenly across SPEs and blocking DMA accesses can be interspersed with computation in the loop. With double buffering, the DMA requests used to fetch data for the first

iteration and commit data to main memory after the last iteration, cannot be overlapped with computation. We capture the effect of blocking and non-overlapped DMA as:

$$O_{spe} = T_{rec} + T_{sen} + T_c + T_{DMA} \quad (5)$$

The last term in Equation 5 is itemized to the blocking DMAs performed within loop iterations and the non-overlapped DMAs exposed when the loop is unrolled, tiled, and executed with double buffering. We use static analysis of the code to capture the sizes of DMA transfers.

## 4.4. Cluster Execution Modeling

We generalize our model of a single asymmetric multi-core processor to a cluster by introducing an inter-processor communication component as:

$$T = (T_{ppe} + O_{ppe}) + (T_{spe} + O_{spe}) + C \quad (6)$$

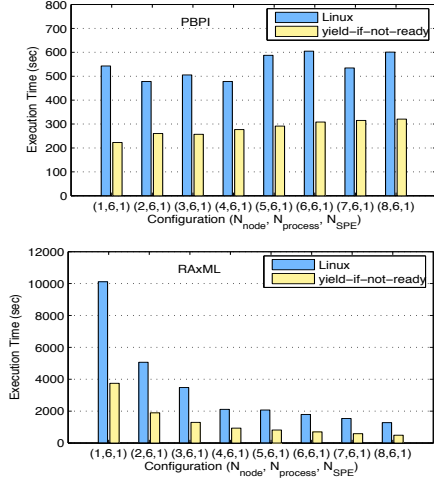
We further decompose the communication term  $C$  in communication latency due to each distinct type of communication pattern in the program, including point-to-point and all-to-all communication. Assuming MPI as the programming model used to communicate across nodes or between address spaces within nodes, we use `mpptest` to estimate the MPI communication overhead for variable message sizes and communication primitives. The message sizes are captured by static analysis of the application code.

## 4.5. Verification

We verify our model by executing PBPI and RAXML on all feasible layered decompositions that use 1 to 6 PPE threads, 1 to 6 SPEs per PPE and up to 8 PS3 nodes. Figure 3(a),(b) illustrates that the model is accurate both in terms of predicting execution time and in terms of discovering optimal application decompositions and mappings for different cluster scales and data sets. The optimal decomposition varies across multiple dimensions, including application characteristics, such as granularity of off-loaded tasks, frequency and size of communication and DMA operations, size and structure of the data set used in the application, and number of nodes available for execution. Accurate modeling of the application under each scenario is valuable to tame the complexity of discovering the optimal decomposition. In our test cases, the model achieves error rates consistently under 15%. The mean error rate is 5.2%. The model predicts accurately the optimal configuration and mapping in all 48 test cases.

## 5 Co-Scheduling on Asymmetric Clusters

Although our model projects optimal mappings of MPI applications on the PS3 cluster with high accuracy, it is



**Figure 6.** Performance of yield-if-not-ready policy and the native Linux scheduler in PBPI and RAXML.  $x$ -axis notation:  $N_{node}$  - number of nodes,  $N_{process}$  - number of processes per node,  $N_{SPE}$  - number of SPEs per process.

oblivious to the implications of user-level and kernel-level scheduling on oversubscribed cores. More specifically, the model ignores cases in which PPE threads and SPE threads are not co-scheduled when they need to synchronize through shared memory.

The main objective of co-scheduling in the context of heterogeneous multi-core processors is to minimize slack time on SPEs, since SPEs bear the brunt of the computation in practical cases. This slack is minimized whenever a thread off-loaded to an SPE needs to communicate or synchronize with its originating thread at the PPE and the originating thread is running on a PPE hardware context.

As illustrated in Figure 4, different scheduling policies can have a significant impact on co-scheduling, slack, and SPE utilization. In Figure 4(a), PPE threads are spinning while waiting for the corresponding off-loaded threads to return results from SPEs. The time quantum allocated to each PPE thread by the OS can cause continuous mis-scheduling with respect to SPE threads.

In Figure 4(b), the user-level scheduler uses a yield-if-not-ready policy, which forces each PPE thread to yield the processor, whenever a corresponding off-loaded SPE thread is pending completion. This policy can be implemented at user-level by having PPE threads poll shared-memory flags that matching SPE threads set upon completion. Figure 6 illustrates the performance of this policy in PBPI and RAXML, when the PPE is oversubscribed with 6 MPI processes, each off-loading on 1 SPE. The results show that compared to a scheduling policy which is oblivious to PPE-SPE co-scheduling, yield-if-not-ready achieves a performance improvement of 1.7–2.7 $\times$ . Yield-if-not-ready bounds the slack by the time needed to con-

text switch across  $p - 1$  PPE threads,  $p$  is the total number of active PPE threads, but can still cause temporary mis-scheduling and slack, as shown in Figure 4(c). Figure 4(d) illustrates an adaptive spinning policy, in which a thread either spins or yields the processor, based on which thread is anticipated to offload the soonest on an SPE. This policy uses a prediction which can be derived with various algorithms, the simplest of which is using the execution length of the most recently off-loaded task from any given thread as a predictor of the earliest time that the same thread will be ready to off-load in the future. The thread spins if it anticipates that it will be the first to off-load, otherwise it yields the processor.

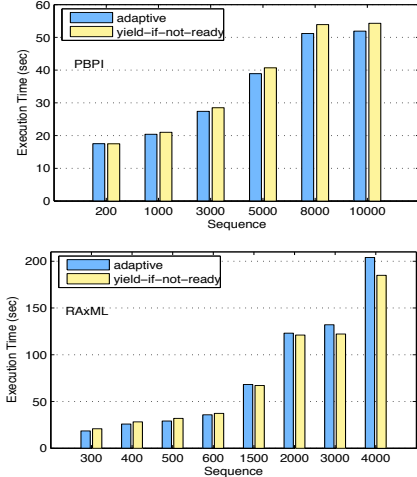
Although the aforementioned adaptive policy can reduce accelerator slack compared to the yield-if-not-ready policy, it is still suboptimal, as it may mis-schedule threads due to variations in the execution lengths of consecutive tasks off-loaded by the same thread, or variations in the run lengths between any two consecutive off-loads on a PPE thread. We should also note that better policies –with tighter bounds on the maximum slack–, can be obtained if the user-level scheduler is not oblivious to the kernel-level scheduler and vice versa. Devising and implementing such policies is a subject of ongoing research.

Figure 7 illustrates results when PBPI and RAXML are executed with various co-scheduling policies. Both applications are executed with variable sequence length ( $x$ -axis), hence variable SPE task sizes. In PBPI, Figure 7(a), adaptive spinning outperforms yield-if-not-ready in all cases. In RAXML, Figure 7(b), adaptive spinning performs better for small data sets, while yield-if-not-ready performs better for large data sets. In RAXML, the variance in length of the off-loaded tasks increases with the size of the input sequence, causing more mis-scheduling when the adaptive policy is used. In PBPI, the task length is not varying, which enables nearly optimal co-scheduling by the adaptive spinning policy. In general, the best co-scheduling algorithm can improve performance by more than 10%. We emphasize that the optimal co-scheduling policy changes with the dataset, therefore support for flexible co-scheduling algorithms in system software is essential on the PS3 cluster.

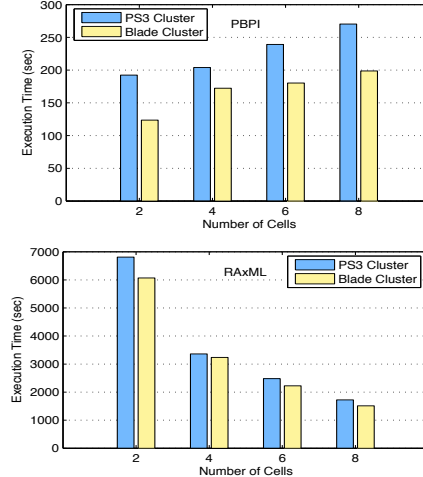
## 6 PS3 versus IBM QS20 Blades

We compare the performance of the PS3 cluster to a cluster of IBM QS20 dual-Cell/BE blades. Our model has been deployed on the QS20 cluster, and has demonstrated accuracy similar to that observed on the PS3 cluster.

The Cell/BE processors on the QS20 have 8 active SPEs and possibly other undisclosed microarchitectural differences. Although both the QS20 cluster and the PS3 cluster use GigE, communication latencies tend to be markedly lower on the QS20 cluster, first due to the absence of a hy-



**Figure 7.** Performance of different scheduling strategies in PBPI and RAXML.



**Figure 8.** Comparison between the PS3 cluster and an IBM QS20 cluster.

pervisor, which is a communication bottleneck on the PS3 cluster, and second due to exploitation of shared-memory communication between two Cell/BE processors on each QS20, instead of one Cell/BE processor on each PS3.

We present selected experimental data points where the two platforms use the same number of Cell processors. On the QS20 cluster, we use both Cell processors per node. Figure 8 illustrates execution times of PBPI and RAXML on the two platforms. We report the execution time of the most efficient pair of application configuration and co-scheduling policy, on any given number of Cell processors. We observe that the performance of the PS3 cluster is reasonably close (within 14% to 27% for PBPI and 11% to 13% for RAXML) to the performance of the QS20 cluster. The difference is attributed to the reduced number of active SPEs and slower communication on the PS3 cluster,

Interestingly, if we compare datapoints with the same total number of SPEs (48 SPEs on 8 PS3's versus 48 SPEs on 6 QS20's), in RAXML the PS3 cluster outperforms the QS20 blade. This result does not indicate superiority of the PS3 hardware or system software, as we apply experimentally defined optimal decompositions and scheduling policies on both platforms. It rather indicates the implications of layered parallelization. Oversubscribing the QS20 with 8 MPI processes (versus 6 on the PS3) introduces significantly higher scheduling overhead and brings performance below that of the PS3. This result stresses our earlier observations on the necessity of models and better schedulers for asymmetric multi-core clusters.

## 7 Related Work

In the context of programming models and supporting environments, recent contributions such as Sequoia [10],

Cell SuperScalar [3], CorePy [12] and PPE-SPE code generators from single-source modules [9, 14], address the problem of achieving high performance with reduced programming effort. While the aforementioned programming models provide interesting alternatives for developing code on systems featuring one Cell processor, our work is targeted at clusters of Cell processors. We address issues related to the optimization of the runtime execution environment in the presence of both distributed memory (MPI) and shared memory (DMA) communication, and the exploitation of layered task and data parallelism on Cell/BE clusters running MPI codes. We believe that supporting layered Cell/BE parallelization within MPI codes provides a graceful migration path for the most dominant parallel programming paradigm in use today. Also, our contributions on modeling and co-scheduling on the Cell/BE are generic enough to be applied to the runtime layer of any of the aforementioned experimental programming models.

A recent evaluation of MPI point-to-point communication and a blocked matrix-matrix multiplication algorithm on a small (4-node) cluster of PS3 nodes is the contribution closest to the work presented in this paper [2]. We depart from this work by conducting an analysis of a PS3 cluster on a larger scale, by focusing on the impact of asymmetric layered parallelism on point-to-point and collective communication, and by presenting models and scheduling techniques for asymmetric parallelism on clusters.

Our contribution overlaps with recent research on developing and optimizing the MPI library for intra-processor communication on Cell [15]. We consider the implications of scaling MPI codes and the point-to-point and collective MPI communication primitives across Cell processors in our modeling framework, however we do not consider Cell-specific optimizations of MPI collectives or point-to-point

communication protocols. Our modeling framework also generalizes existing models of architectural aspects of the Cell/BE, such as the DMA engine [8], by considering intra-Cell and inter-Cell parallel execution.

## 8 Conclusions

We evaluated a very low-cost HPC cluster based on PlayStation3 consoles and proposed a model of asymmetric parallelism and software support for orchestrating asymmetric parallelism extracted from MPI programs on the PlayStation3 cluster. While PS3 has several limitations as an HPC platform, including limited storage and limited support for advanced networking, it has enough computational power compared to vastly more expensive multi-processor blades and forms a solid experimental testbed for research on programming and runtime support for asymmetric multi-core clusters. The model presented in this paper accurately captures heterogeneity in computation and communication substrates and helps the user or the runtime environment map layered parallelism effectively to the target architecture. The co-scheduling heuristics presented in this paper increase parallelism and minimize slack on computational accelerators. Future work will explore co-scheduling support within the Linux kernel and integration of our modeling and scheduling framework with high-level Cell/BE programming models.

## Acknowledgment

This research is supported by the NSF (grants CCF-0346867, CCF-0715051, CNS-0521381, CNS-0720673, CNS-0709025, CNS-0720750), the DOE (grants DE-FG02-06ER25751, DE-FG02-05ER25689), and by IBM through an IBM Faculty Award (grant VTF-874197). We thank Alexandros Stamatakis, Xizhou Feng, and Kirk Cameron for providing us with the original MPI implementations of PBPI and RAXML and for discussions on modeling the Cell/BE. We thank Georgia Tech, its Sony-Toshiba-IBM Center of Competence, and NSF, for the Cell/BE resources that have contributed to this research.

## References

- [1] S. Alam, R. Barrett, J. Kuehn, P. Roth, and J. Vetter. Characterization of scientific workloads on systems with multi-core processors. In *Proc. of IEEE International Symposium on Workload Characterization (IISWC)*, 2006.
- [2] Alfredo Buttari and Jakub Kurzak and Jack Dongarra. Limitations of the Playstation 3 for High Performance Cluster Computing. Technical Report Technical Report UT-CS-07-597, Innovative Computing Laboratory, University of Tennessee Knoxville, Apr. 2007.

- [3] P. Bellens, J. M. Pérez, R. M. Badia, and J. Labarta. Memory – CellSs: a programming model for the cell BE architecture. In *Proc. of Supercomputing'2006*, page 86, 2006.
- [4] F. Blagojevic, X. Feng, K. Cameron, and D. S. Nikolopoulos. Modeling Multi-grain Parallelism on Heterogeneous Multi-core Processors: A Case Study with the Cell BE. In *Proc. of the 2008 HiPEAC Conference*, Jan. 2008.
- [5] F. Blagojevic, D. Nikolopoulos, A. Stamatakis, and C. Antonopoulos. Dynamic Multi-grain Parallelization on the Cell Broadband Engine. In *Proc. of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 90–100, Mar. 2007.
- [6] F. Blagojevic, A. Stamatakis, C. Antonopoulos, and D. Nikolopoulos. RAXML-CELL: Parallel Phylogenetic Tree Construction on the Cell Broadband Engine. In *Proc. of the 21st International Parallel and Distributed Processing Symposium*, Mar. 2007.
- [7] L. Chai, Q. Gao, and D. K. Panda. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. In *Proc. of CC-Grid2007*, May 2007.
- [8] T. Chen, Z. Sura, K. M. O'Brien, and J. K. O'Brien. Optimizing the Use of Static Buffers for DMA on a CELL Chip. In *Proc. of the 19th International Workshop on Languages and Compilers for Parallel Computing*, pages 314–329, 2006.
- [9] A. E. Eichenberger et al. Using Advanced Compiler Technology to Exploit the Performance of the Cell Broadband Engine Architecture. *IBM Systems Journal*, 45(1):59–84, 2006.
- [10] K. Fatahalian, D. R. Horn, T. J. Knight, L. Leem, M. Houston, J. Y. Park, M. Erez, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: Programming the Memory Hierarchy. In *Proc. of Supercomputing'2006*, page 83, 2006.
- [11] W. Gropp and E. Lusk. Reproducible Measurements of MPI Performance Characteristics. In *Proc. of the 6th European PVM/MPI Users Group Meeting*, pages 11–18, Sept. 1999.
- [12] C. Mueller, B. Martin, and A. Lumsdaine. CorePy: High-Productivity Cell/B.E. Programming. In *Proc. of the First STI/Georgia Tech Workshop on Software and Applications for the Cell/B.E. Processor*, June 2007.
- [13] J. A. Turner. Roadrunner: Heterogeneous Petascale Computing for Predictive Simulation. Technical Report LANL-UR-07-1037, Los Alamos National Lab, Las Vegas, NV, Feb. 2007. ASC Principal Investigator Meeting.
- [14] A. L. Varbanescu, H. J. Sips, K. A. Ross, Q. Liu, L.-K. Liu, A. Natsev, and J. R. Smith. An Effective Strategy for Porting C++ Applications on Cell. In *Proc. of the 2007 International Conference on Parallel Processing*, page 59, 2007.
- [15] M. Velamati, A. Kumar, N. Jayam, G. Senthilkumar, P. Baruah, S. Kapoor, R. Sharma, and A. Srinivasan. Optimization of Collective Communication in Intra-Cell MPI. In *Proc. of the 14th IEEE International Conference on High Performance Computing (HiPC), Lecture Notes in Computer Science 4873*, pages 488–499, 2007.