Exploring Many-Core Architecture Design Space for Parallel Discrete Event Simulation

Yi Zhang, Jingjing Wang, Dmitry Ponomarev, and Nael Abu-Ghazaleh
Department of Computer Science, Binghamton University
Binghamton, NY, USA
yzhang25@binghamton.edu, jwang36@cs.binghamton.edu,
dima@cs.binghamton.edu, nael@cs.binghamton.edu

ABSTRACT

As multicore and manycore processor architectures are emerging and the core counts per chip continue to increase, it is important to evaluate and understand the performance and scalability of Parallel Discrete Event Simulation (PDES) on these platforms. Most existing architectures are still limited to a modest number of cores, feature simple designs and do not exhibit heterogeneity, making it impossible to perform comprehensive analysis and evaluations of PDES on these platforms. Instead, in this paper we evaluate PDES using a full-system cycle-accurate simulator of a multicore processor and memory subsystem. With this approach, it is possible to flexibly configure the simulator and perform exploration of the impact of architecture design choices on the performance of PDES. In particular, we answer the following four questions with respect to PDES performance and scalability: (1) For the same total chip area, what is the best design point in terms of the number of cores and the size of the on-chip cache? (2) What is the impact of using in-order vs. out-of-order cores? (3) What is the impact of a heterogeneous system with a mix of in-order and out-of-order cores? (4) What is the impact of object partitioning on PDES performance in heterogeneous systems? To answer these questions, we use MARSSx86 simulator for evaluating performance, and rely on Cacti and McPAT tools to derive the area and latency estimates for cores and caches.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event, Parallel*

General Terms

Design

Keywords

PDES, Multi-cores, full-system simulation

SIGSIM-PADS'14, May 18–21, 2014, Denver, CO, USA.

http://dx.doi.org/10.1145/2601381.2601392.

1. INTRODUCTION

Parallel Discrete Event Simulation (PDES) is a fine-grained application with irregular communication patterns and frequent synchronization. For these reasons, its performance and scalability have been severely constrained in computing platforms with high communication delays. The continuing emergence of multi-core and many-core architectures offers a significant promise with respect to PDES performance. Several recent studies evaluated the performance impact of multi-core chips on PDES [2, 14, 15, 7, 31]. However, these studies are limited to the existing hardware platforms with modest number of cores per chip [2, 14], or to specialized chips with larger core counts, such as the Tilera Tile64 architecture [15]. Therefore, while these prior works are still useful and more such studies are likely to emerge as the new multi-core processors are developed and brought to market, they can only answer a limited set of questions, and it is unclear whether the conclusions of these studies can be easily generalized beyond specific platforms and organizations being investigated.

In this paper, we expand the scope of PDES investigations on multi-core and many-core systems by employing a simulation-based approach. Specifically, instead of executing PDES on real platforms, we study its behavior using a cycle-accurate full-system simulator of a multi-core system. Simulation-based studies are extensively used by the computer architecture community where standard and emerging applications are evaluated against future hypothetical designs of single-core, or multi-core, processors and memory hierarchies. A major advantage of such an approach is that a simulator can be flexibly configured and the impact of a wide range of design options (most of which are not currently available in real designs, but could appear in some form in future processors) can be evaluated. In this framework, PDES serves as a benchmark of interest which we evaluate using the MarSSx86 simulator [27]. Marss is a recently developed cycle-accurate simulator of x86-based multi-core processors and memory hierarchy.

To the best of our knowledge, this is the first study that evaluates PDES on emerging architectures (rather than on existing architectures). Such simulation-based approach allows us to evaluate a much wider range of systems and explore the issues related to the architecture impact on PDES performance and scalability that are otherwise impossible to evaluate using traditional measurements on existing systems. Furthermore, using circuit, timing and area analysis tools, we study and compare PDES performance for various chip designs with the equivalent area, taking into account

the area used by the cores and the on-chip cache hierarchy. Alternatively, we can also express possible performance and scalability improvements in PDES as a function of the additional chip area needed to achieve these improvements.

Specifically, this paper focuses on the following explorations and addresses the following questions:

First, we evaluate the trade-off between increasing the core count on a chip and using larger on-chip last-level cache. The optimal breakdown between these two resources depends on the applications running on the system. While placing more cores on a chip would allow for a higher degree of parallelism without having to cross the chip boundary, larger caches reduce expensive off-chip memory accesses. To understand this tradeoff in the context of PDES, we study the performance of PHOLD executing under ROSS simulator [6] on simulated hardware systems with several core/cache configurations ranging from the design where most of the chip area is dedicated to processing cores to the design where most of the area is dedicated to the on-chip caches.

Second, we evaluate the impact of the individual core designs on the performance of PDES. Specifically, we consider the choice between large out-of-order superscalar cores, smaller out-of-order cores, and simple in-order cores. In scenarios where fast event processing is on the critical path, more complex cores can offer a significant performance boost. However, since PDES is often dominated by communication, synchronization and rollback overhead, it is also conceivable that in-order cores will not result in significant overall slowdown for many models. At the same time, the use of simpler and smaller cores allows to place more of them on the same chip for the same total chip area, thus increasing the low-latency parallelization opportunities; our study considers these trade-offs.

Third, we investigate PDES performance issues on a heterogeneous many-core system that is composed of a number of high performance out-of-order cores and a number of small in-order cores. A particular question that we investigate on these systems with respect to PDES is whether or not a reconsideration of PDES object-to-core partitioning algorithms is required to adjust to the system heterogeneity. While previous work only considered evenly-sized partitions, we evaluate scenarios where a larger number of objects (and therefore, more work) is assigned to more powerful cores.

All simulation-based studies presented in this paper have been conducted using MARSS [27], a cycle accurate full system x86-64 architecture simulator. To obtain area estimates for various core designs with private caches, we used McPAT tool [21], so that all our designs are compared for the same total area. We obtained the latencies for differently-sized last-level caches using CACTI 5.3 [29] tool. The PDES workloads are based on classical PHOLD [10] models.

The key results and conclusions of our experiments are the following:

- For an area-unconstrained chip where the Level 3 (L3) cache size and the number of cores is fixed, and only the core counts vary, the best core choice is the modestly aggressive 2-way superscalar out-of-order core. For the simple PHOLD models, these cores perform nearly identical to much more aggressive 4-way out-of-order superscalar cores, but at the same time they significantly outperform simple in-order cores.

- For an area-constrained chip, a larger number of simple cores results in higher performance compared to the design with smaller number of more powerful cores for PHOLD models. That is, the advantages of thread-level parallelism that can be extracted from multiple cores on the same chip outweigh the advantages of instruction-level parallelism available within each core. Specifically, we show that the best performance is obtained when the largest possible number of small in-order cores are used.

- The size of the shared L3 cache has a limited impact on simulation performance (for PHOLD models) and it is more advantageous for performance to increase the number of cores at the expense of smaller last-level cache. This is because a significant locality in private L1 and L2 caches make the L3 cache less critical for performance. This conclusion is true even for the variation of PHOLD that intentionally increases the memory pressure by introducing an array manipulation loop inside each event.

- Heterogeneous multi-cores are detrimental to PDES performance, as the synchrony of the simulation progress on multiple (heterogeneous) cores is naturally distorted, resulting in the loss of efficiency. However, the degree to which performance is impacted depends on the composition of the system (e.g. the nature and number of individual cores), and some designs have only modest performance degradation. In general, the performance is limited by the slowest cores in the system, thus wasting the extra processing capabilities of more powerful cores.

- A heterogeneity-aware PDES object partitioning may somewhat alleviate the performance challenges of heterogeneous designs, but our initial results show that naive partitioning modifications (such as doubling the number of objects placed on the more powerful cores) only lead to further performance problems. Future research is needed to determine the proper partitioning for such systems.

The rest of the paper is organized as follows. Section 2 provides PDES background of the key PDES features relevant to this study. Section 3 describes our evaluation methodology and tools used for evaluating performance, area and latency. Section 4 presents the results of our experiments and provides discussion of these results. Section 5 describes the related work and we conclude in Section 6.

2. PDES BACKGROUND

In Discrete Event Simulation (DES), a model consists of a set of simulation objects with associated state [8, 20, 34]. For example, in a logic simulation, the objects may be the different gates in the model. The simulation begins with a number of scheduled events. For example, the initial list of scheduled events may be a set of test vectors presented at specified times to the inputs of the circuit. Events (containing timestamps that denote when the event is to take effect) are ordered by simulation time in a pending event queue. Simulation proceeds by processing the event with the earliest timestamp, which can cause changes in the simulation state (*e.g.*, by changing the state of a gate) and schedule

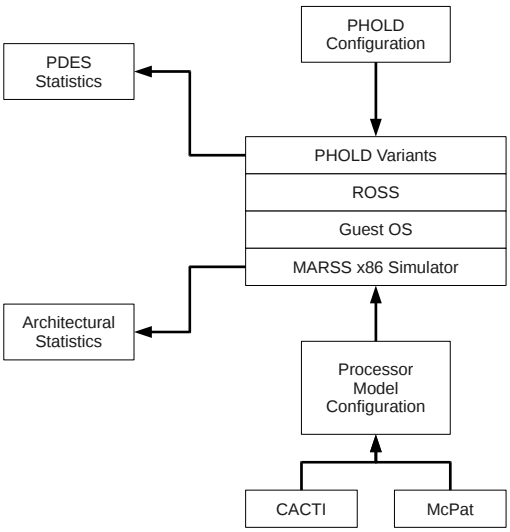


Figure 1: Evaluation and Simulation Methodology

one or more future events (*e.g.*, changing the value at the input of gates connected to the gates whose output logic level changed). Simulation time then advances to the timestamp of the next pending event. Simulation terminates when there are no more events or when a predetermined simulation state is reached.

Parallel Discrete Event Simulation (PDES) leverages parallel processing to accelerate the performance and capacity of DES [11]. The simulation model is partitioned across multiple simulation processes (called Logical Processes, or LPs). Each LP maintains a local pending event queue and carries out simulation as described above, repeatedly processing the earliest time stamp event. A locally processed event may generate events to remote LPs (that is, affecting state changes of an object managed by a remote LP). Thus, LPs communicate by exchanging time-stamped event messages [3, 9, 23, 28]. Correct simulation requires that all events be processed in their causal order such that parallel execution produces simulation results that are consistent with serial execution. Therefore, a synchronization model is needed to ensure that remote events are processed in their proper causal order.

Optimistically synchronized PDES simulators (that we consider in this study) do not enforce causality during event processing. When an event is received with an earlier processing time than the current simulation time, a causality error is detected [18]. The error is recovered from by rolling back the simulation to an earlier state and canceling event messages that were sent prematurely. Optimistic synchronization potentially hides the latency of communication by allowing LPs to process speculatively instead of waiting for events. However, optimism can lead to a number of problems as the speculative computation can generate premature events that later prove to be erroneous, leading to cascading (propagating) rollbacks [11].

PDES is historically difficult to parallelize: it is a fine grained application with frequent communication and complex dynamic dependency patterns. These characteristics can limit the ability of a parallel simulator to exploit the naturally available parallelism in models. Event processing is typically of low computational complexity, as each event updates the state of an object and possibly schedules future events. Thus, significant communication occurs as events are generated to other objects. The amount of communication across cores depends on the locality of event messages. If event messages are frequently exchanged with remote LPs, significant communication occurs, placing substantial pressure on the memory subsystem of a multi-core processor. In the PHOLD benchmark that we use for this study, the amount of remote communication can be explicitly controlled. Additionally, as the degree of parallelism increases, contention for the use of the shared queues starts to play an important role.

3. EVALUATION METHODOLOGY

A number of tools and simulators were used in this study. Our modelling framework is summarized in Figure 1.

In order to perform cycle-accurate simulation of PDES, we used MARSSx86 [27] - a full-system simulator for x86 multi-core architectures. MARSS models different types of processing cores, including both in-order and out-of-order designs, and also provides flexible configurations for the on-chip cache hierarchy.

In order to estimate the area requirements of the individual cores, we used McPAT tool [21]. The three types of cores that we considered in this study include: a) large out-of-order core (Large OoO); b) small out-of-order core (Small OoO), and c) small in-order core. The configurations of each of these cores are listed in Table 1.

For the cache area, we used a cache byte equivalent area (CBE) model to estimate the size of the shared cache from available area [13]. Under this model, the cache size S can be calculated from A_{avail}, the available area and A_{unit}, the area of a fixed unit size of cache, as shown in Equation 1.

$$S = \frac{A_{avail}}{A_{unit}} \tag{1}$$

We used 4.75 sq. mm per MB as the unit area for the shared L3 cache. For different cache sizes, we derived cache

Table 1: Core Model Configurations

	Large OoO	Small OoO	Small In-order
Issue Width	6	3	2
Commit Width	4	3	N/A
RoB Size	168	64	N/A
Instruction Queue Size	32	32	16
ALU	6	3	1
FPU	6	3	1
Load Queue Size	48	24	N/A
Store Queue Size	96	24	N/A
Private L1-I Cache Size	32KB	32KB	32KB
Private L1-D Cache Size	32KB	32KB	32KB
Private L2 Cache Size	256KB	256KB	256KB
Core Size (sqmm)	19.6154	13.0023	5.2573

access time estimates from Cacti 5.3 tool [29]. We used 32nm technology node for both CACTI and McPAT.

As a PDES simulation engine, we used ROSS-MT [14], a multi-threaded optimistic simulator specifically designed for multi-core environments. ROSS-MT is based on ROSS (Rensselaer's Optimistic Simulation System) [6], a PDES simulation engine with support for both conservative and optimistic time warp simulations.

Typically, PDES performance studies on the real systems are performed using PHOLD benchmark [10]. PHOLD is simple but effective synthetic model for testing the performance of the simulation system. In basic PHOLD, the simulation model is a collection of Logical Processes (LPs). Each Processing Element (PE) is allocated an equal number of LPs. Each LP is initialized with the same number of initial events. The simulation progresses by each LP randomly selecting a target and scheduling an event to that target. When the target receives an event, it randomly selects another LP and schedules an event. At any time during the simulation, the total event population is kept the same. For this study, we utilized PHOLD, but extended it with two additional parameters. The first parameter controls percentage of events that are generated remotely (i.e. to a simulation object running on a different core). The second parameter controls the event processing computational granularity (EPC), which reflects the amount of time that the CPU spends processing each event.

Using the modelling components described above, we performed evaluations in the following directions:

- A study of area-unconstrained homogeneous systems with fixed size of shared cache and variable number of cores.
- A study of area-constrained homogeneous systems, with a trade-off between the size of the on-chip cache and the number of cores. This includes the evaluation of PHOLD models with high memory pressure.
- A study of area-constrained heterogeneous systems composed of multiple types of cores.

For all area-constrained experiments, we assumed 220 square mm chip area for cores and caches, similar to Intel's Ivy-Bridge design [17]. The clock rate is modeled at 3.2 GHz for all of the experiments. Within that constraint, we vary the number and type of processing cores and the amount of shared on-chip L3 cache, to determine the design point that provides the best performance for PDES applications.

Our area-unconstrained model assumes an L3 shared cache of 16 MB, with the cache hit time of 17 cycles. Such a cache occupies 76 sq. mm of area under the CBE model. The configuration of ROSS that we used for most of our experiments is shown in Table 2. All alterations of this configuration are explicitly described whenever they are used.

Table 2: ROSS Configuration for Evaluating Area-Unconstrained Homogeneous Systems

Model	PHOLD
Synchronization Method	Optimistic
Total LPs	24000
Simulation End Time	16
EPC	100
Remote Event Percentage	10%
Memory Pressure	None

3.1 Cache/Core Area Tradeoff

For evaluating the area tradeoffs, we conduct the experiments with increasing core count. Each core has a private L1 and L2 cache, which are kept fixed and included in the core area. The shared L3 cache size is calculated based on the available area remaining for it, using the CBE model mentioned above. CACTI 5.3 is used to derive the latency estimates for the considered L3 cache sizes. Tables 3 and Table 4 provide details of the cache sizes used in our experiments, their corresponding latencies (in cycles) were derived from CACTI.

3.2 Modelling High Memory Pressure

In an attempt to stress-test the core-cache trade-offs, we designed a modified PHOLD model that increases the memory pressure. This was achieved by adding several memory operations during each event processing. Specifically, we used event-private circular buffers, such that every circular buffer contains an array of pointers to a number of discrete blocks in memory. In the event handler, the contents of each block are copied to another block (not contiguous with the source block) within the same circular buffer. This involves a series of memory operations which affects a memory area of $BlockSize \times BlocksPerCircularBuffer$ in size for each event. The configuration of ROSS used for this experiment is shown in Table 5.

Table 5: Configuration of ROSS for Area-Constrained Homogeneous System with Memory Pressure

Model	PHOLD
Synchronization Method	Optimistic
Total LPs	24000
Simulation End Time	16
EPC	100
Remote Event Percentage	10%
Memory Pressure ($BlockSize \times BlocksPerBuffer$)	$256B \times 32, 128B \times 16$

3.3 Evaluating Heterogeneous Processors

For this evaluation, we only consider systems with two different types of cores. Using the three types of core models described above, we formed three possible combinations of cores:

- Large Out-of-order + Small Out-of-order

Table 3: Core Counts and Cache Size (Out-of-order cores)

Core#	Large OoO		Small OoO	
	L3 Cache Size (Byte)	L3 Cache Latency (Cycle)	L3 Cache Size (Byte)	L3 Cache Latency (Cycle)
2	39905280	23	42826048	24
3	35575104	22	39956224	23
4	31244992	19	37086464	23
5	26914816	18	34216640	22
6	22584640	17	31346880	19
7	18254528	16	28477056	19
8	13924352	13	25607296	18
9	9594176	11	22737536	17
10	5264064	7	19867712	17
11	933888	5	16997952	16
12	N/A	N/A	14128128	13
13	N/A	N/A	11258368	12
14	N/A	N/A	8388608	11
15	N/A	N/A	5518784	7
16	N/A	N/A	2649024	6

Table 4: Core Counts and Cache Size (Inorder cores)

Core#	L3 Cache Size (Byte)	L3 Cache Latency (Cycle)	Core#	L3 Cache Size (Byte)	L3 Cache Latency (Cycle)
3	45088768	24	23	21909696	17
5	42770816	24	25	19591808	17
7	40452928	23	27	17273856	16
9	38135040	23	29	14955968	13
11	35817088	22	31	12638080	12
13	33499200	20	33	10320192	11
15	31181312	19	35	8002240	8
17	28863424	19	37	5684352	7
19	26545472	18	39	3366464	7
21	24227584	18	41	1048576	6

- Large Out-of-order + Inorder
- Small Out-of-order + Inorder

As these experiments focus on the tradeoff between different types of cores, we fixed the shared L3 cache size to the 16 Megabytes cache with a latency of 17 cycles. This results in the remaining area of 144 sq.cm on the processor chip usable for the cores. The core combinations (with the number of cores of each type) used in our experiments are shown in the Table 6.

Table 6: Combinations of Cores Used in Evaluations of Heterogeneous Chips

Large OoO + Small OoO		Large OoO + Inorder		Small OoO + Inorder	
Large OoO	Small OoO	Large OoO	Inorder	Small OoO	Inorder
1	9	1	23	1	24
2	8	2	19	2	22
3	6	3	16	3	20
4	5	4	12	4	17
5	3	5	8	5	15
6	2	6	5	6	12
				7	10
				10	2

As the cores in a heterogeneous processor would have different capability, we also implemented workload partitioning for heterogeneous systems in an attempt to bridge the gap between the core processing capability and its actual load. The partitioning is performed by mapping different number of Logical Processors(LPs) to different cores using a specific ratio. With such partitioning strategy, cores with more computing power will receive more LPs, while the overall number of LPs would stay the same.

For example, for a heterogeneous processor with two small out-of-order cores and one in-order core, if the overall number of LPs is 2500 and the partitioning ratio is 2:1, then the two out-of-order cores will receive 1000 LPs each, and the in-order core will receive 500 LPs.

In this experiment, we tested 3 different partitioning ratios, as shown in Table 7.

Table 7: ROSS Configuration for Evaluating Area-Constrained Heterogeneous Systems

Model	PHOLD
Synchronization Method	Optimistic
Total LPs	24000
Simulation End Time	16
EPC	100
Remote Event Percentage	10%
Memory Pressure	NONE
Partitioning Ratio	1:1, 2:1, 4:1

4. PERFORMANCE EVALUATION

In this section we present performance evaluation of ROSS-MT using MARSSx86 simulator. In particular, we first evaluate the tradeoff between the L3 cache size and the core counts, and its impact on the performance of ROSS-MT. We follow this by evaluating the performance of ROSS-MT in the system with heterogeneous cores and examining the impact of object partitioning among the cores on the performance.

4.1 The Impact of L3 Cache Size and Core Counts

In our first experiment, we evaluated the performance of ROSS-MT using large out-of-order cores (large OoO), small out-of-order cores (small OoO), and small in-order cores (in-order) respectively, as shown in Figure 2. In this experiment, we fixed the size of the L3 cache at 16 MBytes. As shown in Figure 2, the performance of ROSS-MT scales when more cores are used. In addition, the performance of simulation when running on large out-of-order cores is close to the one running on small out-of-order cores, but is better than the performance obtained with in-order cores. This indicates that for the PHOLD model, a large out-of-order core has a similar processing speed with small out-of-order core, but is significantly faster than an in-order core. Thus, if the number of cores on the chip is fixed, the best performing design is the one that uses modestly aggressive out-of-order cores.

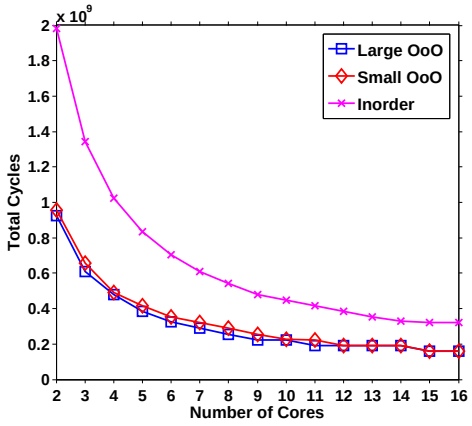


Figure 2: Experiments with Fixed L3 Cache Size

In the next experiment, we evaluate the performance of ROSS-MT when the L3 cache size varies. In this experiment, the chip area was fixed, meaning that the available size of the L3 cache becomes smaller, as more cores are placed on the chip. The size of the L3 cache and the core counts are calculated by Equation 1. Figure 3(a) and Figure 3(b) show the total cycles and the L3 hit rate respectively when the core count increases. At the point where 41 in-order cores are used (the maximum number of in-order cores that could fit on the chip according to our area model), PDES performance is better compared to using either 11 large out-of-order cores or 16 small out-of-order ones (the maximum number of out-of-order cores fitting in the same area).

In addition, the available size of the L3 cache decreases as the core count increases, thus reducing the L3 hit rate, as shown in Figure 3(b). As the penalty of a L3 cache miss cannot be ignored, we expect that the L3 cache misses can dominate the PDES performance at some point when the size of L3 cache is small. However, the experimental results in Figure 3 indicate that the performance of ROSS-MT is always better with more cores, even if the memory pressure exerted by each processing event is increased, as shown in Figure 4. The reason is that due to the high locality in the local L1 and L2 caches, the absolute number of accesses to L3 is fairly small, so the L3 performance has a limited impact in the simulation models that we consider for this study.

4.2 PDES on Heterogeneous Multi-Cores

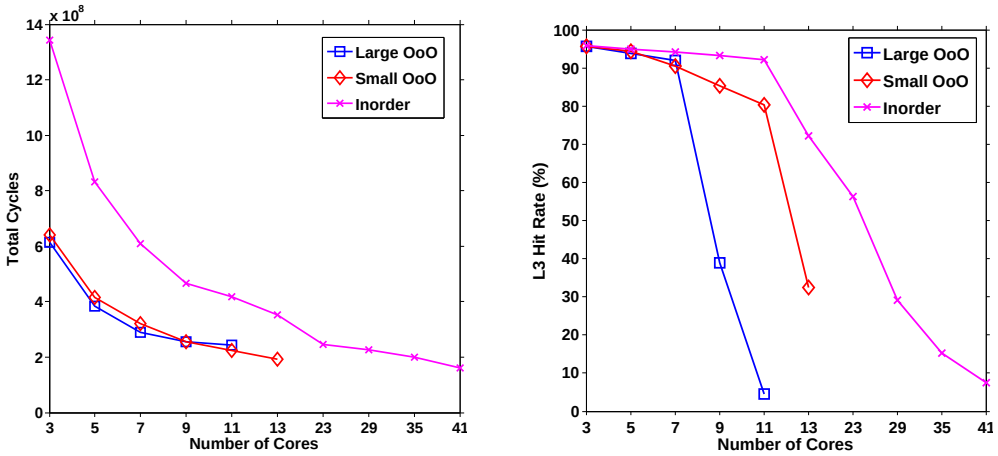
In this subsection, we evaluate the impact of heterogeneous core combinations on the performance of ROSS-MT. In particular, we consider three different core compositions: large OoO + small OoO (Figure 5), large OoO + Inorder (Figure 6), and small OoO + inorder (Figure 7). In Figure 5, Figure 6, and Figure 7, the x-axis N x T indicates that the combination consists of N larger cores and T smaller ones. For example, the case marked "3x6" in Figure 5, refers to the design with 3 large out-of-order cores and 6 small out-of-order cores.

To study the performance of ROSS-MT on heterogeneous cores, a partitioning ratio was introduced in the previous section, with the purpose of distributing objects between larger cores and smaller ones. The motivation is to partially hide the impact of heterogeneity by assigning a larger fraction of objects to the more powerful cores. For example, when the partitioning ratio is set to four, the number of objects assigned to a larger core is four times more than those assigned to a smaller core.

We used the simulation efficiency to evaluate the quality of partitioning. In PDES, efficiency is calculated by dividing committed events to the total processed events. If objects are not properly partitioned among the cores, the progress of simulation on an overloaded core will lag behind, thus causing a large number of rollbacks (assuming an optimistic simulation). Figure 5(b), Figure 6(b) and Figure 7(b) show the simulation efficiency in each group of heterogeneous cores respectively. We selected three values of partitioning ratio: 1, 2, and 4. We observed that the power of the in-order cores (in terms of instruction throughput) is roughly half that of the out-of-order cores, and therefore the partitioning ratio of 2 represents an attempt to directly compensate for this disparity in order to achieve a more synchronous execution. Partitioning ratio of 4 overloads the fast cores even more.

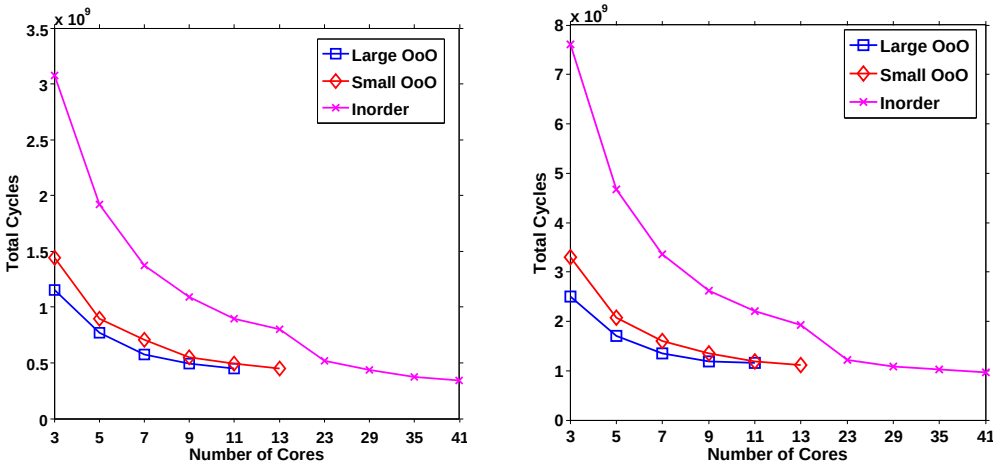
When the partitioning ratio is set as either 2 or 4, the efficiency increases with the number of larger cores. This indicates that larger cores become overloaded , and are more likely to cause the simulation on the smaller ones to be rolled back. When the number of larger cores increases, the likelihood of communication occurring between larger cores and smaller ones reduces, thus increasing the efficiency. However, of these three partitioning ratios, the best efficiency is achieved for the ratio of 1, indicating that objects in the PHOLD model should be equally distributed among cores even though cores are heterogeneous. Therefore, a naive partitioning that matches the number of objects to the core's throughput capabilities does not work. It is likely that some fractional partitioning (the ratio being between 1 and 2) would lead to a better performance than that of equal partitioning, but additional experiments are needed to validate that claim.

We also evaluated the number of cycles for the PDES simulation in each group of heterogeneous cores, as shown in Figure 5(a), Figure 6(a), and Figure 7(a) respectively. In the case of partitioning ratio of 4, the performance of simulation in each experiment becomes better when more larger cores are used. Such performance improvement is obtained by improving the simulation efficiency. On the other hand, the trends are different when the partitioning ratio is set as either 1 or 2. For example, Figure 5(a) shows the number of cycles for simulation when both large out-of-order and small out-of-order cores are used. As both types



(a) Total Cycles (b) L3 Cache Hit Rate

Figure 3: Tradeoff between L3 Cache Size and Core Counts



(a) Small Memory Pressure (*BlockSize*=128, *BlocksPerCircularBuffer*=16) (b) Large Memory Pressure (*BlockSize*=256, *BlocksPerCircularBuffer*=32)

Figure 4: The Impact of Memory Pressure

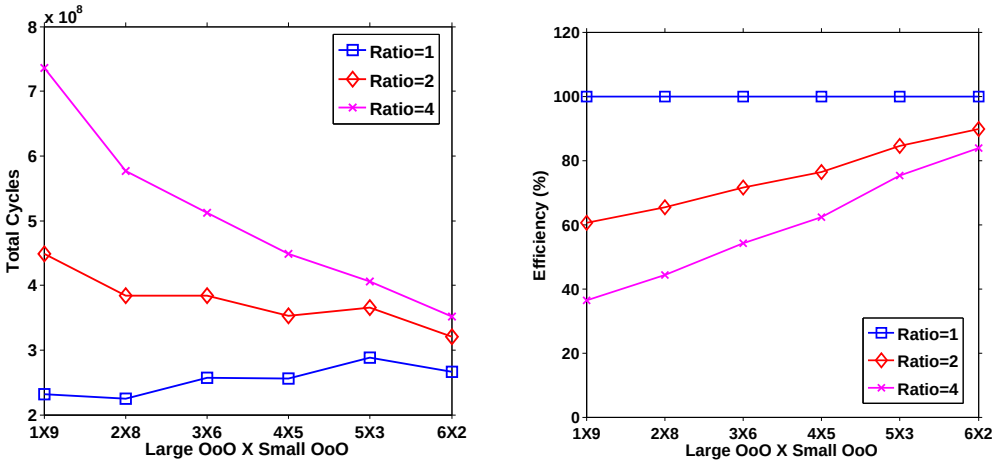
of cores have a similar processing speed, the performance of simulation is mainly dependent on the total number of cores being used when the partitioning ratio is set as 1. In other words, the more cores are used, the better performance can be achieved. In the case of partitioning ratio of 2, the performance of simulation is dominated by the rollbacks.

Finally, Figure 6(a) and Figure 7(a) shows the number of cycles for ROSS-MT when the system is composed of large out-of-order cores and in-order cores, and small out-of-order cores and in-order cores respectively. In both experiments, when the partitioning ratio is set as either 1 or 2, the performance of simulation becomes worse when the number of in-order cores in the composition reduces. As we described in the previous subsection, it is advantageous to use more in-order cores when the size of chip area is limited. Again, notice that partitioning ratio of 1 provides the best performance out of these three points, and additional experiments are needed to determine the best-performing partitioning ratio.

5. RELATED WORK

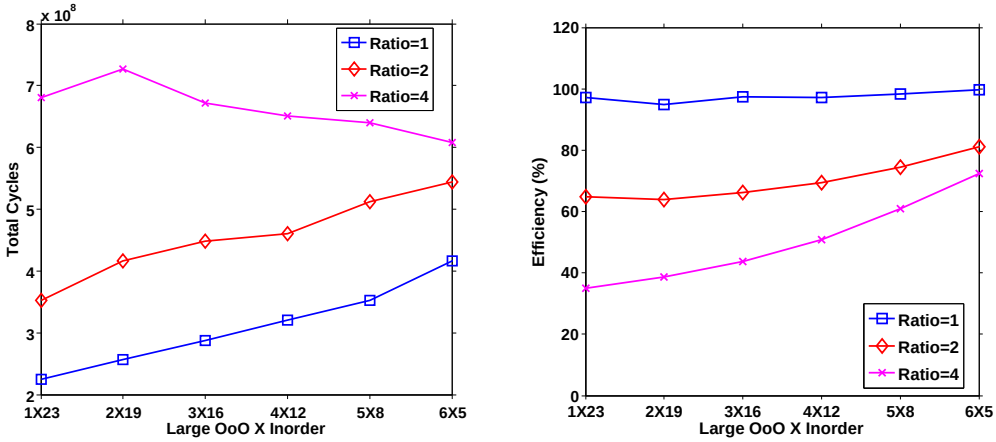
The multi-core and many-core processor architectures can substantially reduce the communication cost of PDES. Several prior works analyzed PDES performance on these emerging systems and also explored performance optimization opportunities. Jagtap et al. [14, 15] designed a multi-threaded version of ROSS simulator, called *ROSS-MT*, to explicitly exploit shared memory hierarchy available on multi-core systems. This is in contrast to the process-based communication model used in baseline ROSS simulator, and in many other PDES engines. The performance of ROSS-MT was evaluated on two emerging many-core platforms: 48-core AMD Opteron Magny-Cours [14] and Tilera Tile64 [15]. The experimental results showed that ROSS-MT can scale well on both platforms, especially when a number of performance optimizations (specific to each platform) are applied.

Vitali et al. [32, 31] developed a different multi-threaded PDES simulator on multi-core platforms. A load-sharing scheme was implemented, allowing each simulation kernel instance to be executed by multiple threads. Wilsey et al. [7]



(a) Total Cycles (b) ROSS Efficiency

Figure 5: Core Composition between Large OoO and Small OoO



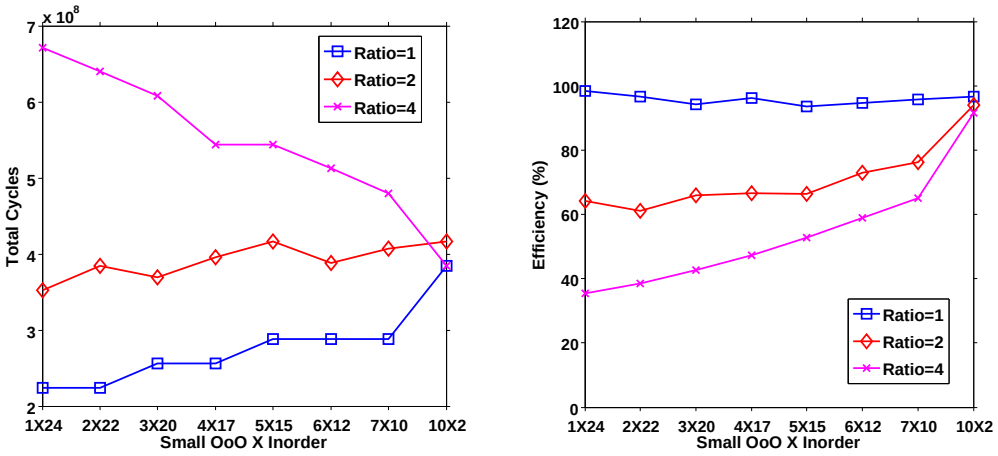
(a) Total Cycles (b) ROSS Efficiency

Figure 6: Core Composition between Large OoO and Inorder

improved the performance of Time Warp simulation, by dynamically changing the frequency of each core during the simulation. In this approach, the cores having LPs with fewer rollbacks are overclocked, while the cores containing LPs with more rollbacks are underlocked.

Several studies focused on investigating the design trade-offs in multi-core architectures in order to improve the area efficiency of these systems. A study of [13] is an example of CMP design space exploration in terms of selecting different core designs (in-order and out-of-order) and also investigating the trade-offs between the core count and the cache capacity. In this paper, we perform a similar study in the context of multi-threaded PDES. Several studies focused on developing analytical models for the core, cache and on-chip interconnect area [1, 26, 33, 19]. The cache subsystem design has also been studied in great detail. For example, last level cache designs for data mining applications were explored in [16]. The work of [35] evaluated the impact of CMP cache sharing on multi-threaded applications and demonstrated benefits of cache-sharing aware program transformations. Oh et al. [26] and Wentzlaff [33] proposed simple analytical models to study trade-offs between the core count and the cache capacity under finite die area. Analytical models for thermal implications of CMP designs [24] and its impact on network scalability [4] have also been developed.

The performance of parallel systems, however, depends not just on the CMP design but also on the characteristics of the workload [5]. Some studies have explored such interplay. Configuration workload characterization [25] demonstrated that architectural configuration alone or workload characterization alone were not sufficient to predict the performance of the system. The work of [12] developed analytical models for both architecture and workloads and studied the applications performance and power implications on a range of architectures. Lotfi et al. proposed Scale-out processor design [22] to address the inefficiency of current architectures (which are designed for high single thread performance) to achieve high server throughput. They proposed a chip architecture based on pods - an optimal performance-density building block specifically for scale-out workloads. The work by Vitali et al. [30] considered the memory access patterns



(a) Total Cycles (b) ROSS Efficiency

Figure 7: Core Composition between Small OoO and Inorder

for various operations within optimistic simulation. They demonstrated a cache-aware memory manager that maps the simulation data structures in a way that maximizes the cache performance.

6. CONCLUSIONS

We presented a simulation methodology for evaluating performance of PDES applications on emerging hardware many-core architectures. Our approach allows for the exploration of the design space, including the evaluation of the designs with large number of cores as well as the designs with heterogeneous cores. To the best of our knowledge, ours is the first work that uses PDES as an application for such simulation-based studies.

The conclusions of our initial experiments with this simulation framework are that the shared L3 cache has a minimal performance impact for PHOLD-style PDES applications, and it is more advantageous for performance to utilize the available chip area for additional cores rather than caches. While modestly aggressive out-of-order cores provide noticeable performance benefits compared to the in-order cores for area-unconstrained designs, a better performance is achieved with a larger number of simple in-order cores when the chip area is constrained. We also demonstrated that core heterogeneity negatively impacts PDES performance as it naturally distorts the synchrony, increases the number of rollbacks and degrades simulation efficiency. We also showed that specific performance impact depends on the composition of cores.

The simulation framework described in this paper can also be applied to a number of more detailed studies, such as examining and understanding the causes and the impact of rollbacks in PDES in a repeatable and controlled fashion. The low-level details of other PDES subsystems can also be studied using this methodology - this is left for our future work. Future work will also examine further modifications to PDES object partitioning algorithms to tailor them to heterogeneous platforms and mitigate the negative impact of core heterogeneity on PDES performance.

7. ACKNOWLEDGEMENTS

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-11-2-0004 and by National Science Foundation grants CNS-0916323 and CNS-0958501. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies and endorsements, either expressed or implied, of Air Force Research Laboratory, National Science Foundation, or the U.S. Government.

8. REFERENCES

[1] O. Azizi, A. Mahesri, S. Patel, and M. Horowitz. Area-efficiency in cmp core design: co-optimization of microarchitecture and physical design. *ACM SIGARCH Computer Architecture News*, 37(2):56–65, 2009.

[2] K. Bahulkar, N. Hofmann, D. Jagtap, N. B. Abu-Ghazaleh, and D. Ponomarev. Performance evaluation of PDES on multi-core clusters. In *Proceedings of the 2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications, (DS-RT 10)*, pages 131–140, 2010.

[3] M. L. Bailey, J. V. Briner, Jr., and R. D. Chamberlain. Parallel logic simulation of VLSI systems. *ACM Computing Surveys*, 26(3):255–294, Sept. 1994.

[4] J. Balfour and W. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 187–198. ACM, 2006.

[5] M. Bhadauria, V. Weaver, and S. McKee. Parsec: hardware profiling of emerging workloads for cmp design. In *Proceedings of the 23rd international conference on Supercomputing*, pages 509–510. ACM, 2009.

[6] C. D. Carothers, D. Bauer, and S. Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648 – 1669, 2002.

[7] R. Child and P. Wilsey. Dynamically adjusting core frequencies to accelerate time warp simulations in

many-core processors. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 35–43. IEEE, 2012.

[8] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[9] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[10] R. Fujimoto. Performance of time warp under synthetic workloads. 1990.

[11] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Interscience, Jan. 2000.

[12] Z. Guz, O. Itzhak, I. Keidar, A. Kolodny, A. Mendelson, and U. Weiser. Threads vs. caches: modeling the behavior of parallel workloads. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pages 274–281. IEEE, 2010.

[13] J. Huh, D. Burger, and S. Keckler. Exploring the design space of future cmps. In *Parallel Architectures and Compilation Techniques, 2001. Proceedings. 2001 International Conference on*, pages 199–210. IEEE, 2001.

[14] D. Jagtap, N. Abu-Ghazaleh, and D. Ponomarev. Optimization of parallel discrete event simulator for multi-core systems. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 520–531. IEEE, 2012.

[15] D. Jagtap, K. Bahulkar, D. Ponomarev, and N. Abu-Ghazaleh. Characterizing and understanding pdes behavior on tilera architecture. In *Workshop on Principles of Advanced and Distributed Simulation (PADS 12)*, July 2012.

[16] A. Jaleel, M. Mattina, and B. Jacob. Last level cache (llc) performance of data mining workloads on a cmp-a case study of parallel bioinformatics workloads. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 88–98. IEEE, 2006.

[17] D. James. Intel ivy bridge unveiled – the first commercial tri-gate, high-k, metal-gate cpu. In *Custom Integrated Circuits Conference (CICC)*, pages 1–4, 2012.

[18] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):405–425, July 1985.

[19] R. Kumar, V. Zyuban, and D. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, pages 408–419. IEEE, 2005.

[20] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 3rd edition, 2000.

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 469–480, New York, NY, USA, 2009. ACM.

[22] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, et al. Scale-out processors. In *Proceedings of the 39th International Symposium on Computer Architecture*, pages 500–511. IEEE Press, 2012.

[23] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, Mar. 1986.

[24] M. Monchiero, R. Canal, and A. González. Design space exploration for multicore architectures: a power/performance/thermal view. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 177–186. ACM, 2006.

[25] H. Najaf-Abadi and E. Rotenberg. Configurational workload characterization. In *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*, pages 147–156. IEEE, 2008.

[26] T. Oh, H. Lee, K. Lee, and S. Cho. An analytical model to study optimal area breakdown between cores and caches in a chip multiprocessor. In *VLSI, 2009. ISVLSI'09. IEEE Computer Society Annual Symposium on*, pages 181–186. IEEE, 2009.

[27] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*, 2011.

[28] P. F. Reynolds Jr. A spectrum of options for parallel simulation. In *Winter Simulation Conference*, pages 325–332. Society for Computer Simulation, 1988.

[29] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. Jouppi. Cacti 5.3. *HP Laboratories, Palo Alto, CA*, 2008.

[30] R. Vitali, A. Pellegrini, and G. Cerasuolo. Cache-aware memory manager for optimistic simulations. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, pages 129–138. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012.

[31] R. Vitali, A. Pellegrini, and F. Quaglia. Assessing load-sharing within optimistic simulation platforms. In *Proceedings of the 2012 Winter Simulation Conference*. IEEE, 2012.

[32] R. Vitali, A. Pellegrini, and F. Quaglia. Towards symmetric multi-threaded optimistic simulation kernels. In *Principles of Advanced and Distributed Simulation (PADS)*, pages 211–220. IEEE, 2012.

[33] D. Wentzlaff, N. Beckmann, J. Miller, and A. Agarwal. Core count vs cache size for manycore architectures in the cloud. 2010.

[34] B. P. Zeigler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press Inc. (London) Ltd., 24/28 Oval Road, London NW1, 1984.

[35] E. Zhang, Y. Jiang, and X. Shen. Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs? In *ACM Sigplan Notices*, volume 45, pages 203–212. ACM, 2010.