Characterizing and Understanding PDES Behavior on Tilera Architecture

Deepak Jagtap, Ketan Bahulkar, Dmitry Ponomarev, Nael Abu-Ghazaleh
Computer Science Department
State University of New York at Binghamton
{djagtap1, kbahulkar, dima, nael}@cs.binghamton.edu

***Abstract*—The emergence of manycore architectures with shifting balance between computation and communication overhead can have a tremendous impact on performance and scalability of fine-grained parallel applications such as PDES. It may also be necessary to rethink the design philosophy of key PDES subsystems, that were traditionally focussed on hiding long communication delays.**

In this paper, we perform extensive evaluation of PDES on Tile64Pro - a new 64-core chip from Tilera. For our studies, we use the recently developed multithreaded version of the popular ROSS simulator and show that the performance of this simulator (with many optimizations proposed) scales by a factor of 27X when it is executed on 56 cores of the Tilera chip for Phold benchmark with 20% remote communication. We also evaluate the impact of performance optimizations that we propose on both conservative and optimistic versions of the simulator and also analyze the sensitivity to various simulation parameters. Finally, we explore the issues of object placement and model partitioning on Tilera architecture.

I. Introduction

Performance aspects and bottlenecks of Parallel Discrete Event Simulation (PDES) have been extensively studied for traditional cluster computing environments, where long communication latencies across the cluster elements pose significant challenges to PDES scalability on these platforms. Recently, continuing emergence of multicore processors and the potential of these architectures to drastically minimize the impact of communication-related issues motivated several studies that examine performance and scalability of PDES in these environments [1], [2]. For example, the work of [2] demonstrated the advantages of using multithreaded (as opposed to multiprocess) implementation of ROSS simulator [3] on a quad-core Intel Core i7 machine and on a AMD Magny-cours system [4] composed of four 12-core chips for the total of 48 cores. Several optimizations were also proposed that allowed almost 3x performance improvement for the multithreaded version over baseline MPI-based implementation of ROSS [2].

These previous studies were limited to the architectures with modest numbers of cores per chip, which is representative of what is available on the market today. However, as the degree of integration continues to increase with smaller transistor feature sizes and newer fabrication technologies, current multicore architectures can soon be replaced by many-core designs with the number of cores per chip in tens, hundreds or even thousands [5], [6]. In fact, some researchers believe that the number of cores available on a chip will double every 18 months, calling it a "new Moore's Law" [6]. While these many-core chips are not yet widely available, some examples already exist. For example, Intel recently announced the prototype of an 80-core chip [5]. Another example is the 64-core Tilera Tile64 chip [7] that utilizes a tiled CPU and cache architecture and employs a two-dimensional mesh network as an interconnection fabric between the cores. We view the Tilera architecture as an example of a future manycore chip, and the main goal of this paper is to study PDES performance on this platform. We believe that this paper represents the first study of its kind and the lessons learned will be generally useful for PDES implementations on future manycores systems.

Compared to mainstream multicore processors and their clusters, the Tilera architecture has a number of unique features that have direct impact on performance and scalability of PDES (or any other fine-grain parallel application in general). First, a significantly higher degree of core integration allows a larger number of parallel threads to communicate efficiently without leaving the chip boundaries, thus creating potential for better scalability. Second, the Tilera architecture features a more balanced communication-computation infrastructure, where the communication bottlenecks are significantly reduced and computation cycles emerge as a more significant bottleneck. The reasons for this are slower processing cores (which increases the fraction of time spent on computation) and well-optimized mesh interconnection network that promotes both low latency and high throughput communication among the cores. These factors have tremendous implications on an application such as PDES, which was traditionally designed with the goal of hiding long communication latencies.

The starting point of our exploration of PDES behavior on the Tilera platform is the multithreaded implementation of ROSS simulator [3] that was developed in a recent work [2]. Multithreaded PDES directly exploits the presence of shared levels of memory hierarchy on the chip (the shared L2 cache in the case of the Tilera) and eliminates delays due to multiple message copying operations and synchronization delays involved in polling of the queues that are inherent in MPI-based implementations. The work of [2] showed that multithreaded implementation significantly outperforms the

MPI-based design on platforms such as Intel Core i7 and AMD Magny-cours. In this paper, we demonstrate that a multithreaded simulator also significantly outperforms the MPI-based version on the Tilera, especially when a number of performance optimizations are introduced.

In terms of performance optimizations, we propose to adapt three techniques introduced in [2] such that they exploit the features of the Tilera. We also propose some new optimizations that utilize the APIs available from the Tilera Multicore Components (TMC) library; more details on these are provided in Section 3.

When all optimizations are considered, multithreaded ROSS executing the basic Phold model on 56 cores of the Tilera chip (the maximum number of cores that we could use; the other eight cores are reserved for the OS tasks) achieved a speedup of 27X for Phold benchmark at 20% remote events. This compares to only about 18X speedup achieved by the MPI implementation of ROSS. Next, we study the individual impact of the proposed performance optimizations, both for conservative and optimistic simulation. Finally, we address the issues of object placement and model partitioning in the context of the Tilera TilePro64 platform. Our results demonstrate that while Tilera's mesh network exhibits non-uniform core-to-core latencies, the degree of non-uniformity is minimal. Compounded by the fact that the balanced nature of Tilera architecture makes the applications running on it more tolerant to communication delays to begin with, the minimal non-uniformity in latencies make the PDES performance almost insensitive to the placement strategies (i.e. whether the frequently communicating objects are placed on the nearby or on the distant cores). Furthermore, we demonstrate that the model partitioning strategies that just balance the computational load among the cores (these partitions are much easier to derive), very closely approach the performance of partitioning schemes that try to optimize the number of remote communications (through communication graph mincut), as well as balance the workload. These results are important in that they demonstrate that PDES can exhibit great scalability on the Tilera platform with minimum investment in object placement and partitioning decisions.

In summary, the main contributions and the key results of this paper are the following:

- We demonstrate that the Tilera architecture is an extremely effective platform for supporting scalable PDES applications, especially when the simulation engine is redesigned to take advantage of the shared memory hierarchy. In particular, the multithreaded version of ROSS simulator executing Phold benchmark can achieve up to 27X speedup when executing on 56 cores of Tilera chip for Phold 20% remote communication.
- We show that this level of speedup can be achieved with very simple model partitioning and object placement strategies. Specifically, we demonstrate that the partitioning that just balances the computation workload among the various cores achieved nearly the same level of performance compared to partitioning that minimizes inter-core communication (while also attempting to preserve workload balance as much as possible). Furthermore, we show that PDES on Tilera is practically insensitive to the object placement among the processing cores, as long as the overall computational balance is maintained.
- We study the impact of various performance optimizations proposed in this paper on both conservative and optimistic simulation on Tilera and also explore performance sensitivity to various simulation parameters, such as the GVT interval. For example, our results show that the latency of computing the GVT on Tilera is minimum, and therefore it is advantageous to compute GVT more frequently for optimistic simulation, especially for the models prone to rollbacks.

The remainder of this paper is organized as follows. Section II reviews the Tilera architecture and describes performance optimizations considered in this paper. Section III overviews ROSS simulator and its multithreaded implementation. In Section IV, we present experimental methodology followed by the results of our performance evaluation of ROSS on the Tilera platform. Section V reviews the related work. Finally, Section VI offers our concluding remarks.

In the rest of the paper, we perform detailed evaluation of ROSS-MT (with its optimizations) on the Tilera platform.

II. Tilera Tile64Pro Architecture Overview

TilePro64 is a power-efficient 64-core processor from Tilera. It uses switched, on-chip mesh interconnect providing coherent dynamic distributed cache. The processor chip is comprised of 64 power efficient cores (tiles) arranged in the form of an 8x8 matrix. Tiles are connected by six mesh networks forming tight integration of cores. The cache coherence across the cores and the memory provides efficient and scalable platform for shared memory applications. The role of the mesh network is to move data between cores, memory and I/O providing low latency and high bandwidth.

The iMesh Interconnect consists of two classes of networks: the first class comprises a set of software-visible networks for application-level streaming and messaging, while the second consists of the networks used by the memory system to handle memory requests, exchange cache coherency commands and support high performance shared memory communication. Dedicated Switch Engines are used to implement the iMesh Interconnect, allowing for a complete decoupling of data routing from the Processing Engines. The Switch Engine contains six physical mesh networks. The Static network (STN) switches scalar data between tiles with very low latency. The other five are dynamic networks, which facilitate streaming and packet data transfer among tiles and I/O devices. Of the five

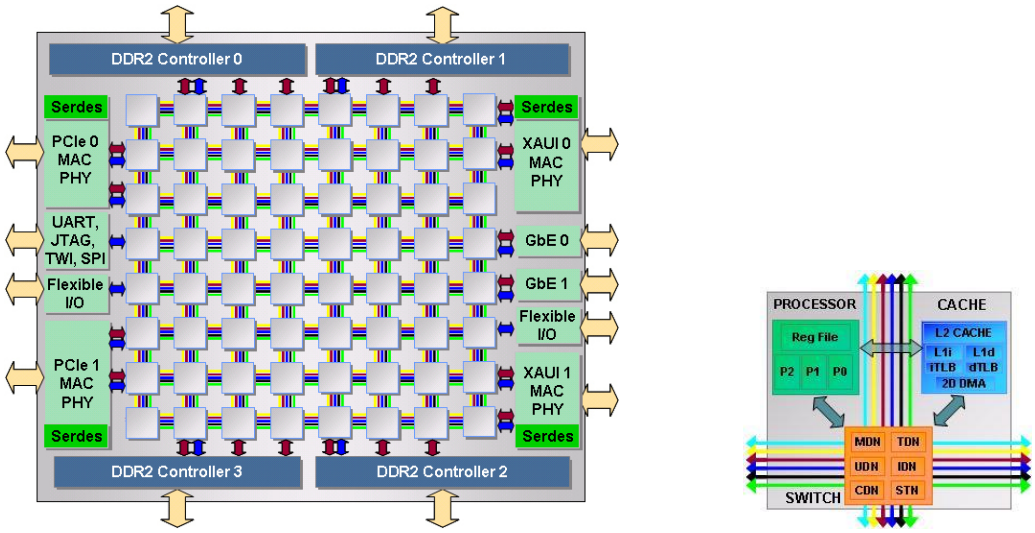

Figure 1. Architecture of the Tilera Processor (used with permission from Tilera Corporation).

dynamic networks, namely the UDN, TDN, MDN, CDN and IDN, only the User Dynamic Network (UDN) is visible to the user. The others are used to satisfy cache misses from external memory and other tiles, for DMA transfers, for I/O, and for various other system-related functions

A single processing tile has a 32-bit 5-stage VLIW pipeline with L1 instruction and data caches, L2 combined data and instruction cache, and a routing engine for the mesh networks. The 64KB L2 caches from each of the cores form a distributed L3 cache accessible by any core and I/O device. Static branch prediction and in-order execution further reduce area and power required. Translation look-aside buffers are present on each core and support memory protection for virtual memory. Each memory controller reorders memory read and write operations to the DIMMs to optimize memory utilization. Cache coherence is maintained by each cache-line having a home core. Upon a miss in its local L2 cache, a core needing that cache-line goes to the home cores L2 cache to read the cache-line into its local L2 cache. Two dedicated mesh networks manage the movements of data and coherence traffic in order to speed the cache coherence communication across the chip. To enable cache coherence, the home core also maintains a directory of cores sharing the cache line, removing the need for power hungry bus-snooping cache coherency protocols. Because the L3 cache leverages the L2 cache at each core, it is extremely power efficient while providing additional cache resources. Figure 1 shows the I/O devices, 10G and 1GB Ethernet, and PCI-e, connecting to the edge of the mesh network. This allows direct writing of received packets into on-chip caches for processing and vice-versa for sending We believe this feature can be exploited by PDES in a clustered environment.

The Tilera platform provides the iLib library which allows parallel programming and provides APIs similar to MPI for message send-receive, all_reduce operation and barrier synchronization primitive. We use MPI library implementation (provided by ISI from University of California Santa Barbara) which acts as a wrapper for iLib APIs and makes MPI application portable on Tilera platforms. iLib internally uses User Data Network (which is iMesh) and provides buffering mechanism for message send-receive.

III. Overview of ROSS Simulator and its Multithreaded Implementation

In this section, we first describe the baseline MPI-based ROSS simulator [3] that we used for this study. Then, we present the design overview of the multithreaded version of ROSS (called ROSS-MT) [2]. Finally, we review the performance optimizations that were previously proposed to accelerate ROSS-MT and discuss their adaptation to Tilera.

In ROSS simulator, the smallest unit of simulation is called Logical Process (LP), which represents an object in the simulation model. Maintaining a single list of all processed events of all LPs within a processing element (PE) results in excessive false rollbacks. In order to reduce false rollbacks, ROSS introduces a notion of Kernel Processes (KPs). LPs are grouped among KPs, rather than grouping all LPs on a single PE. Each PE thus has a number of KPs and each KP in turn refers to a group of LPs. A list of processed events will be associated with each KP instead of PE, this in turn reduces the number of false rollbacks.

Each process (PE) maintains a queue of the outgoing remote events. When an LP sends a message to another remote LP, an event message is first queued in to *Output Queue (Outq)* of the sender PE. Events are then dequeued from Outq and sent to destination PE asynchronously based on the buffer availability. Posted sends and Posted receives buffers are used for asynchronous message passing.

Nonblocking MPI_Isend and MPI_Irecv calls are used for message passing in combination with MPI_Testsome and MPI_Iprobe APIs. Once the event message is successfully received at the destination process, it is queued in to priority queue at the receiver side. The event scheduler is responsible for maintaining priority queue of the pending events and also for performing event processing. To support rollbacks, ROSS uses a reverse computation mechanism (instead of traditional incremental state saving techniques), where each event handler is paired with a reverse computation handler to undo events in case of rollbacks.

A. ROSS-MT Performance Optimizations and their Adaptation to Tilera

To address some of the performance inefficiencies of MPI-based ROSS, we recently implemented its multithreaded version (called ROSS-MT). Here, we present a brief overview of ROSS-MT design and refer the readers to [2] for more details. In ROSS-MT, each thread has a PE associated with it. Furthermore, each thread has its own input queue, a memory manager, an event scheduler and a free event queue for fossil collection. Since all threads share the address space, the activity involved in sending of an event is reduced to simply queueing the pointer of the event to the destination PE. The input queue associated with each PE contains event messages from other PEs. The receiver thread dequeues events from the input queue and inserts them into the event priority queue for processing. The sender thread keeps a copy of each message sent so that it can generate canellation messages from the local copy in case of a rollback.

After careful analysis of the performance bottlenecks in ROSS-MT, we also introduced several performance optimizations [2]. We briefly describe these optimizations below and also outline some modifications that we made to adapt them to the Tilera platform evaluated in this paper.

The first optimization targets efficient cache usage for free memory management. ROSS implements its own free memory management to avoid unecessary use of the memory allocation library. We introduced the Last-In-First-Out (LIFO) approach to message allocation from the free queues. In this scheme, the most recently freed message is used from each free memory sub-pool. This policy improves cache performance and significantly reduces the number of cache misses. For this paper, we also added a new optimization by enabling a thread-specific heap feature available in the Tilera(TMC) library to enhance the local cache usage.

The second optimization introduced in [2] is the distributed locking for the input queue. We observed that on the traditional multicore platforms evaluated in [2] lock contention among the threads for the shared input queue becomes a significant bottleneck. To reduce this contention, the input queue is split into multiple input queues and a group of senders share an input queue. However, maintaining too many queues increases the overhead needed to poll them. Therefore, there is a trade-off between the lock contention overhead (in case of too many threads sharing a queue) and queue polling overhead. Our previous study [2] showed that due to the much higher impact of the lock contention in traditional Intel and AMD multicore systems, the optimal performance was achieved when one queue was used for each sender and receiver (meaning that queue polling overhead was relatively low on those systems).

In contrast, on Tilera we observed that the lock contention is a much lesser issue due to efficient inter-core communication network and that the queue polling overhead (which requires the extra core cycles) is dominant. Therefore, the optimal number of senders sharing a queue needs to be reconsidered, if this optimization is used on Tilera. Specifically, our experiments demonstrate that a single input queue can be shared by eight senders and one reciever without experiencing any lock contention. In order to futher reduce lock-unlock overhead on Tilera, we use the Spin_queued_mutex primitive supported by the Tilera (TMC) library. Spin_queued_mutex are special spin_locks that require a smaller number of cycles compared to pthread_mutex to implement lock and unlock operations.

Finally, the third optimization proposed in [2] targets efficient barrier synchronization. This is important, because barrier synchronization and all-reduce operation are key components for GVT computation - a critical PDES subsystem. ROSS-MT implementation uses its own library for barrier synchronization and all-reduce operation. Our library uses pthread_barrier which uses atomic instructions directly supported by the ISA to optmize barrier operation. We observered that barrier synchronization based on condition variables and pthread_mutex has very high overhead at high degree of parallelism.

In the rest of the paper, we perform detailed evaluation of ROSS-MT (with its optimizations) on the Tilera platform.

IV. Performance Evaluation of ROSS-MT and ROSS-MPI on Tilera

In this section, we present performance evaluation of fully optimized ROSS-MT (multithreaded ROSS) and ROSS-MPI (MPI based ROSS) on Tilera. First, we discuss the evaluation environment, and the simulation benchmark that we use.

A. Experimental Setup and Benchmark

We used the basic Phold benchmark for the initial set of experiments, because it allows us to easily explore a wide range of application characteristics using configurable parameters such as percentage of remote communications, Event Processing Computational granularity (EPC) and the number of objects per PE. We can also control event population by configuring the number of events generated during the initialization of each LP (referred to as start events). We use the above mentioned parameters for evaluating different aspects of scalability of our implementation. Later, to access

the performance impact of various partitioning strategies, we use a more restricted version of Phold, where non-uniform communication patterns are introduced to make the model sensitive to partitioning choices. This model is described later in this section.

B. ROSS-MT Scalability Analysis

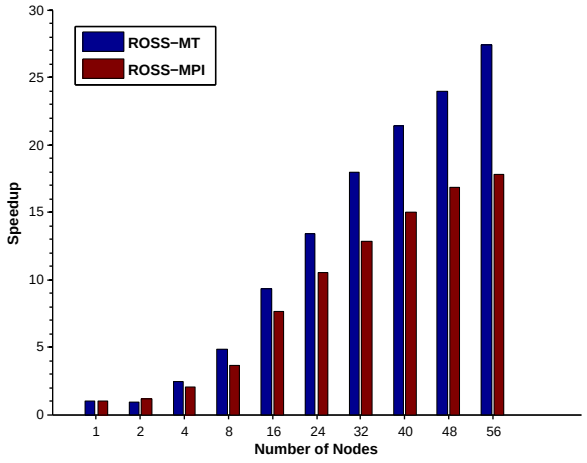

Figure 2. Speedup at 20% Remote Communication

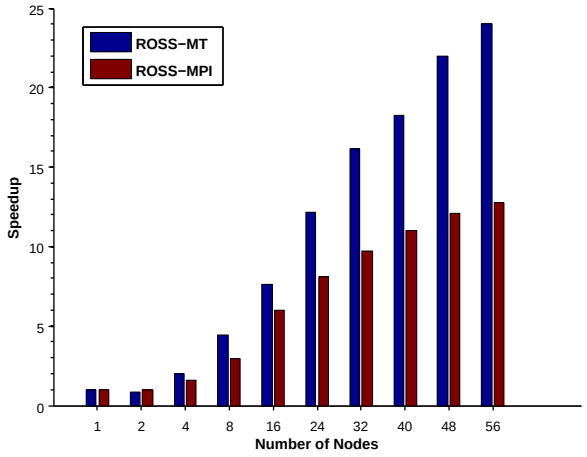

Figure 3. Speedup at 40% Remote Communication

Our first experiment was to study the scalability of the Phold model executed on Tilera as the number of cores used for simulation increases from one (sequential simulation) to 56 (the maximum number of cores available to us). For this experiment, we used 56000 total objects with equal number of objects per PE. We performed the experiments for three different values of remote communication percentage: 20%, 40% and 100%. The baseline for these experiments (the case with 1 core) represents the ROSS simulator running in the optimized sequential mode, without any of the overheads necessary for parallel simulation. The sequential ROSS

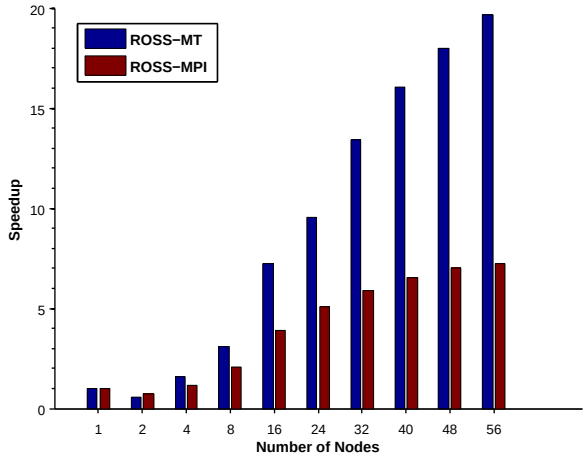

Figure 4. Speedup at 100% Remote Communication

simulator has an option of using a calendar queue or splay tree for the critical event queue [8]; we experimented with both data structures and selected the splay tree because it provided better performance. The sequential simulation runtime was 444 seconds with the splay tree, and 470 seconds with the calendar queue.

As shown in Figures 2, 3 and 4, ROSS-MT is significantly more scalable than ROSS-MPI. For example, for 20% remote events, ROSS-MT exhibits the speedup of 27X at 56-way parallelism, while ROSS-MPI shows the speedup of 18X at a similar setting. For 40% remote events, the respective speedups are 24X and 12X, and for 100% remote events the speedups are still significant, especially for ROSS-MT - 20X and 7X respectively. Note that ROSS-MT generally maintains better scalability trends than ROSS-MPI, the difference between the two increases as the percentage of remote events goes up. The main reason for better scalability on Tilera compared to the traditional multicore architectures is reduced lock contention overhead on Tilera due to the more efficient nature of the communication network. Compared to the MPI implementation, ROSS-MT also saves processing cycles used for message probing in MPI based implementation.

Finally, we observed that the speedup achievable on Tilera is not constrained by the extra pressure on the communication network (and thus higher latencies), but instead is limited by the increased processing delays due to the software overhead of processing remote events. More results demonstrating this effect are presented later.

Next, we evaluate the impact of the performance optimizations for ROSS-MT (described in previous sections) executing on Tilera. These optimizations are driven by the observation that the scalability is limited by three major factors: barrier synchronization, NUMA issues and lock contention on the shared queue. In order to study the importance of each of these optimizations on the Tilera

platform, we did a thorough analysis of each optimization. As discussed in previous sections, we also evaluated the role of PER_THREAD_HEAP optimization. Our analysis are performed for both conservative and optimistic simulation.

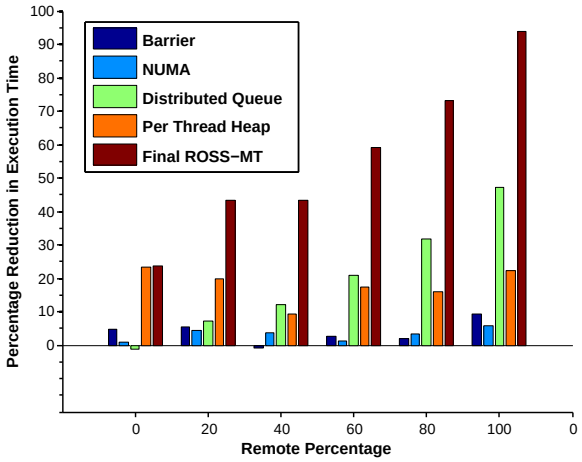

Figure 5. Impact of Optimizations with Increasing Remote Percentage for Optimistic Simulation

Figure 5 depicts the results for the optimistic simulation. Here, NUMA and barrier optimizations play a smaller role. Barrier synchronization implementation based on condition variables and pthread_mutex (as in the baseline ROSS-MT) scales reasonably well on Tilera due to lowered lock contention and a relatively low cost of remote cache access.

Similarly, the NUMA optimization plays a very important role on AMD and Intel platforms [2]. However, since on the Tilera chip all cores and the memory controller are tightly connected in a mesh, the effect of NUMA optimization is significantly smaller. Further, a low clock rate of Tilera cores reduces the mismatch between the CPU speed and the memory access time, thus diminishing the impact of non-uniformity in the memory access latency. Finally, the NUMA optimization also has a negative impact of increased fossil collection overhead. The combination of all these reasons makes NUMA optimization's impact on the overall simulation performance almost negligible.

On the other hand, the distributed queue optimization plays a major role in increasing the scalability and performance of baseline ROSS-MT, with up to 40% reduction in execution time. Our experiments show that 8 PEs can share a single queue without lock contention at 100% remote communication. This implies reduced lock contention as compared to the AMD Mangycours platform. Enabling the PER_THREAD_HEAP feature in the Tilera library reduces memory management overhead by reducing lock contention for a central heap. It also promotes local cache access by placing allocated pages on the same tile.

Figure 6 shows the performance benefits achieved by each optimization for conservative simulation. As shown

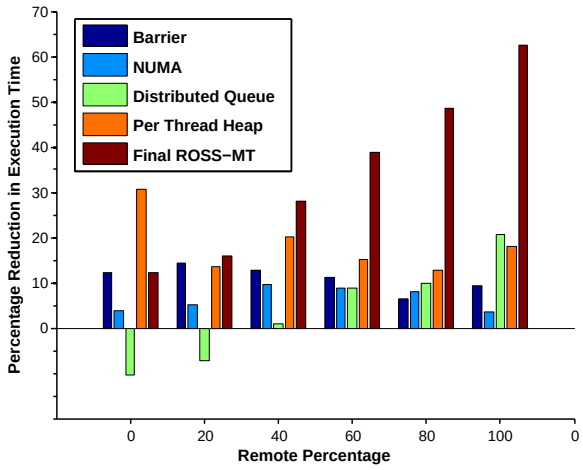

Figure 6. Impact of Optimizations with Increasing Remote Percentage for Conservative Simulation

in this figure, barrier and NUMA optimizations have a noticeable impact. Conservative simulation involves frequent synchronization of PEs, resulting in higher impact of barrier synchronization. Advantages of NUMA optimization come only from the LIFO strategy for conservative simulation. In addition, conservative simulation does not involve fossil collection and thus avoids the negative impact on NUMA-aware optimization. Because of the less frequent access to the input queue, the distributed queue based optimization plays a relatively small role in conservative simulation. At the same time, the use of PER_THREAD_HEAP shows significant improvement for both conservative and optimistic scenarios.

Next, we evaluate performance of ROSS-MT with increased remote communication. For this experiment, we use the basic Phold model in 56-way configuration with 1000 objects per PE. We fix the GVT interval at 256. Execution time is measured at different remote percentage for fixed batch size. We observed that the batch size of 8 is optimal for both multi-threaded ROSS and MPI-based ROSS. We set event processing factor to 0 and thus event processing overhead is limited to creating and sending a new event.

As shown in Figure 7, ROSS-MT significantly outperforms MPI-based ROSS. With the increase in remote communication, the execution time for ROSS-MPI grows linearly. At the same time, the execution time for ROSS-MT increases slightly with increasing remote communication. This behavior can be explained by the fact that additional processing delays introduced for handling remote communications at both sending and receiving nodes dominate the actual wire delays through the iMesh network. For ROSS-MPI, this software overhead of remote message processing and generation is much higher then for ROSS-MT, leading to differences in performance. When all communications are remote (100%), ROSS-MT outperforms ROSS-MPI by

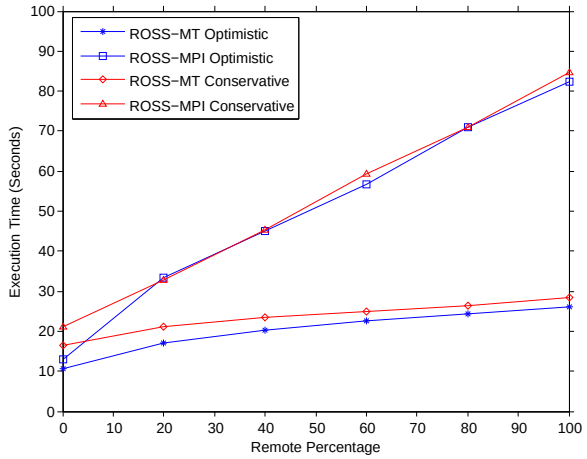

Figure 7. Execution Time for ROSS-MT and ROSS-MPI on Tilera

a factor of more than 3.

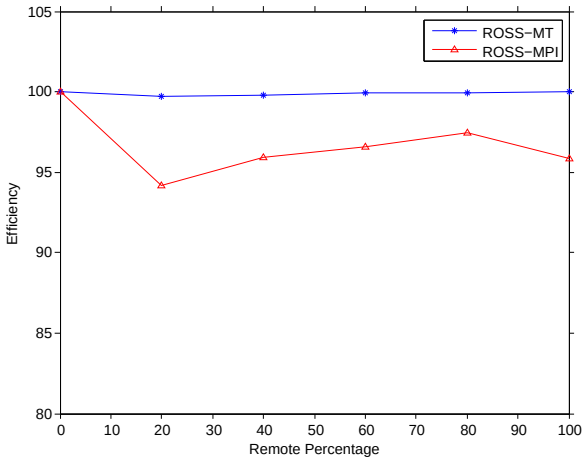

Figure 8. Efficiency at GVT Interval 2048

In the next set of experiments, we analyze the impact of GVT interval on the event processing efficiency. For this study, we gradually increase the GVT interval from 256 to 2048. As shown in Figure 8, efficiency of the ROSS-MT stays noticeably higher than that of ROSS-MPI, even at relaxed GVT synchronization.

Another important observation specific to Tilera is that GVT computation cycle is relatively inexpensive due to the low-latency communication infrastructure. For the model such as basic Phold (that does not experience many roll-backs), this is manifested by the fact that the simulation performance remains relatively constant for different GVT intervals. Thus, performing more frequent GVT computation cycles introduces negligible overhead. Again, this example simply demonstrates that the GVT computation latency is low and computing GVT more frequently is likely to provide advantages especially for models with high rollback probability.

Next, we compare the performance of optimistic and conservative simulation on Tilera. For conservative simulation, the lookahead value is set to 1, while optimistic simulation generates events with time granularity of 1 (similar to lookahead). We set GVT interval to 2048 for this set of experiments. As shown in Figure 7, ROSS-MT outperforms ROSS-MPI by the same factor even for conservative simulation.

C. *Stress-testing the iMesh*

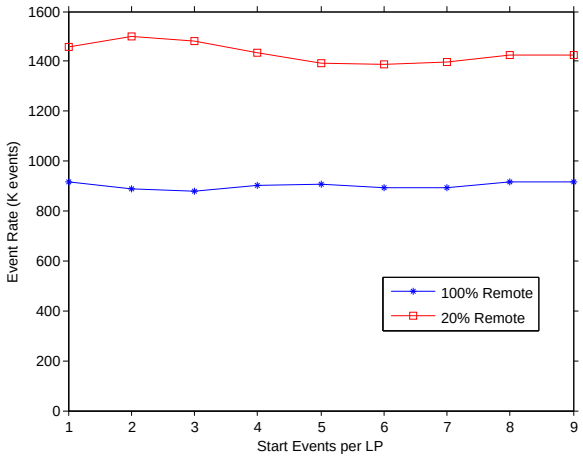

Figure 9. ROSS-MT Performance with Increasing Event Population

The iMesh tile interconnect provides low-latency and high-bandwidth communication among the tiles. The bandwidth of the iMesh interconnect is an important factor affecting the scalability of shared memory applications. Thus, it is important to evaluate ROSS-MT scalability with reference to the iMesh bandwidth. The next experiment that we present is our attempt to saturate the iMesh bandwidth by increasing event population in the Phold model. To this end, we used 56-way Phold simulation with 1000 objects per PE at 100% remote communication, and then gradually increased the event population by varying the number of starting events per LP. As shown in Figure 9, the event rate sustains even for 9 starting events per LP for 100% remote case. Instead of saturating the iMesh, the additional events simply exert the extra pressure on the processing cores, because more core cycles are needed to process the remote events.

The distributed shared cache feature of the Tilera platform plays an important role in determining the performance of highly parallel applications. Thus, it is essential to study the scalability of ROSS-MT with increasing cache pressure by increasing per-tile memory demand. In this experiment, we evaluate the performance of ROSS-MT by gradually increasing the number of LPs per PE to a very high number. As shown in Figure 10 event rate sustains even for 50000 LPs per PE at 100% remote communication. Thus ROSS-MT can

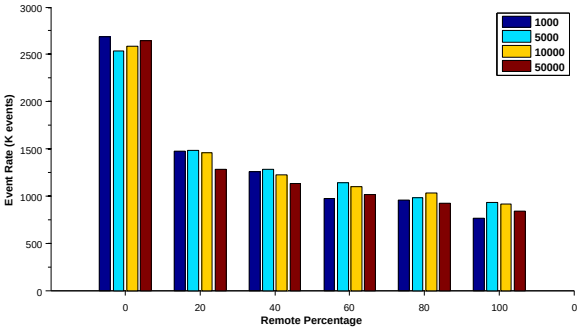

Figure 10. Scalability with Increasing LPs per PE at Different Remote Percentages

scale very well on the Tilera for very large PDES models; performance is only constrained by the computational power of the individual cores.

D. Partitioning and Placement Issues

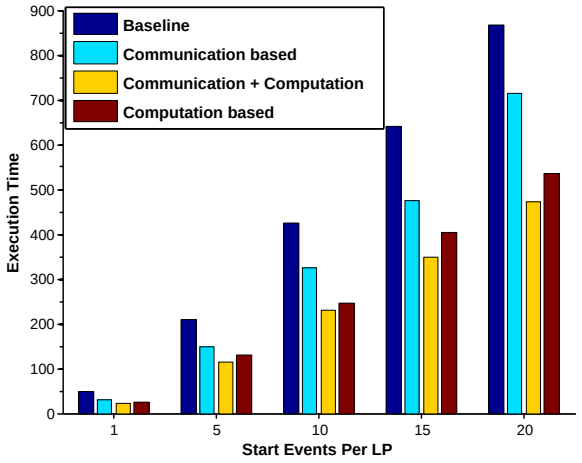

Figure 11. Performance Comparison of Various Partitioning Strategies

Our previous results reported in this section (especially the attempt to saturate the iMesh network) suggest that communication plays a secondary role in defining PDES performance on the Tilera and that the real bottlenecks are the processing cores themselves. We now project this vision into the issue of model partitioning and object placement, which in traditional architectures play an important role for PDES performance. In fact, complex partitioning schemes that accurately balance communication and computation often provide serious performance gains on those systems. In addition, careful object placement is also a must.

To understand the impact of model partitioning while executing PDES on the Tilera, we augmented the Phold model with the capability to define event processing granularity (EPC) for each event. We also overlaid a hierarchical topology on top of the objects such that each object communicates to a fixed set of other objects to make the model sensitive to partitioning (simple random Phold is not). With this enhanced model, we experimented with partitioning schemes that: a) emphasize reduction in the number of inter-core events over load balancing, b) balance computational load without considering the impact on communication; c) equal emphasis on both; and d) randomly distributes objects among the cores to act as a baseline for comparison purposes.

As shown in Figure 11, while the partitioning strategy that takes into account both computation and communication achieves the best performance, the partitioning strategy that emphasizes the computation balance achieves comparable level of performance (within 10% in all cases considered). This result corroborates our previous hypothesis about the computation-dominated nature of the Tilera. The significance of this result is that even a simple partitioning scheme that just balances computation is sufficient to sustain PDES scalability on the Tilera. This kind of partitioning is much easier to obtain than a one that optimizes for communication. For example, the information about event processing is readily available from the model, while communication frequencies have to be obtained through profiling. Furthermore, complex graph partitioning tools are needed to derive communication-optimized partitions. Finally, the same observations apply to dynamic object migration schemes. It is much easier to design them if only the information about the CPU loading needs to be maintained.

Remote %	Nearby placement	Random placement
0	8.00	8.03
20	12.18	12.44
40	14.09	16.64
60	14.96	17.18
80	18.57	19.07
100	21.34	21.98

Table I
PLACEMENT OF PAIRED MODEL

In our next experiment, we evaluate the impact of placement of highly communicating PEs on the performance of ROSS-MT. Due to the mesh topology of tile interconnect, it is logical to place highly communicating PEs on the adjacent tiles for higher performance. In order to evaluate this, we modified the basic Phold, so that communicating pairs of consecutive PEs are formed e.g. (0,1),(2,3). LPs in one PE communicate only with other LPs in the same pair. We measure the execution time of simulation by two PE placement strategies: Nearby and Random. In Nearby placement strategy, we place PEs that form a pair on physically adjacent tiles, while in Random placement PEs in one pair are placed on physically farthest tiles.

As shown in Table I, such placement variations have a negligible impact on the performance, confirming once again

that the iMesh is not a bottleneck and that object placement should be a secondary consideration while running PDES on the Tilera.

V. Related Work

While there were no previous studies of PDES performance on the Tilera platform (to the best of our knowledge), there were studies of time warp performance and scalability on the IBM Blue Gene supercomputer [9], [10]. Just like with Tilera, impressive speedups and performance were achieved [10], again demonstrating the point that PDES can scale very well on these emerging platforms. The key difference between the Blue Gene and Tilera (aside from the obviously different scale of the design) is that the former uses fairly powerful Power series cores, while the latter uses slow energy-efficient cores, thus making computation a dominant bottleneck. The study of [10] also shows that event rate drops by 3X at 100% remote events. On the Tilera, the event rate is sustained even for this high remote percentage.

Previous works have demonstrated the importance of partitioning to reduce the communication frequency in PDES (e.g., [11]). Similarly, dynamic partitioning and workload rebalancing mechanisms have been proposed to repartition the simulation to recover dynamic behavior changes of the simulation model for both conservative (e.g., [12]) and optimistic (e.g., [13]) synchronization protocols.

Fujimoto's GTW simulator is one of the first shared memory optimistic PDES implementations. It exploits shared memory for efficient message communication. GTW also implemented optimizations such as direct cancellation, which allows an LP to cancel out erroneously sent remote events directly, eliminating the need for anti-messages [14]. Similarly, in shared memory, messages can simply be written into a buffer and become visible to all processors. Fujimoto and Hybinette also describe an efficient *on-the-fly* fossil collection algorithm to enable fast reclamation of memory [15]. They also explore efficient buffer management algorithms for shared memory environments [16].

VI. Concluding Remarks

We presented detailed characterization and evaluation of PDES on Tilera Tile64Pro architecture - an example of emerging class of many-core designs. In contrast to traditional communication-dominated parallel computing platforms (for which classical PDES algorithms and techniques were designed and optimized), Tilera represents a computation-dominated environment which has significant implications on PDES.

Specifically, we demonstrated that large speedups can be achieved on Tilera, especially when designs are optimized to take into account the presence of on-chip shared memory hierarchy (as in ROSS-MT). We also demonstrated that PDES optimizations designed for traditional Intel and AMD chips do not necessarily work as such on Tilera and need to be adjusted to take into account the new computation-communication balance. Furthermore, we demonstrated that simple partitioning and dynamic object migration schemes that just take into account the computational balance across the cores provide performance that is competitive (within 10%) of more complex schemes that also optimize for communication. Next, we showed that the object placement across the cores should be a secondary consideration on Tilera, as there is nearly no performance difference between various placement schemes for the models that we considered. Finally, we showed that GVT calculations on Tilera are inexpensive, meaning that frequent GVT cycles can provide optimal performance for optimistic simulation, especially for rollback-prone models.

Acknowledgements

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-11-2-0004. 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 or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. We also gratefully acknowledge support from the National Science Foundation grants CNS-0916323 and CNS-0958501.

References

[1] K.Bahulkar, N.Hofmann, D.Jagtap, N.Abu-Ghazaleh, and D.Ponomarev, "Performance evaluation of pdes on multicore clusters," in *14th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)*, 2010.

[2] D. Jagtap, N.Abu-Ghazaleh, and D.Ponomarev, "Optimization of parallel discrete event simulator for multi-core systems," in *Proc. of IPDPS*, 2012, (forthcoming).

[3] C. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low memory, modular time warp system," in *Proc of the 11th Workshop on Parallel and Distributed Simulation (PADS)*, 2000.

[4] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the amd opteron processor," *IEEE Micro*, vol. 30, no. 2, pp. 16–29, 2010.

[5] "Advancing multi-core technology into the tera-scale era," 2007, available at: http://techresearch.intel.com/articles/Tera-Scale/1449.htm.

[6] K. Asanovic, R. Bodik, J. Demmel, J. Kubiatowicz, K. Keutzer, E. Lee, G. Necula, D. Patterson, K. Sen, J. Shalf, J. Wawrzynek, and K. Yelick, "The landscape of parallel computing research: A view from berkeley 2.0," Jun. 2007, presentation slides available at http://view.ececs.berkeley.edu.

[7] "Tilera TILE64 processor," 2008, documentation from Tilera Website http://www.tilera.com.

[8] R. Rönngren and R. Ayani, "A comparative study of parallel and sequential priority queue algorithms," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 7, no. 2, pp. 157–209, Apr. 1997.

[9] K. Perumalla, "Scaling time warp-based discrete event execution to 10^4 processors on a blue gene supercomputer," in *Proc. of the ACM Conference on Computing Frontiers (CF)*, 2007.

[10] D. Bauer, C. Carothers, and A. Holder, "Scalable time warp on bluegene supercomputer," in *Proc. of the ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2009.

[11] L. Li and C. Tropper, "A design-driven partitioning algorithm for distributed verilog simulation," in *Proc. 20th International Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007, pp. 211–218.

[12] A. Boukerche and S. Das, "Dynamic load balancing strategies for conservative parallel simulation," in *Proc. 11th Workshop on Parallel and Distributed Simulation (PADS)*, 1997, pp. 32–37.

[13] P. Peschlow, T. Honecker, and P. Martini, "A flexible dynamic partitioning algorithm for optimistic distributed simulation," in *Proc. 20th International Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007.

[14] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "GTW: a Time Warp system for shared memory multiprocessors," in *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Eds., Dec. 1994, pp. 1332–1339.

[15] R. M. Fujimoto and M. Hybinette, "Computing global virtual time in shared-memory multiprocessors," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 4, pp. 425–446, 1997.

[16] R. Fujimoto and K. Panesar, "Buffer management in shared-memory Time Warp system," in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS 95)*, Jun. 1995, pp. 149–156.