PROJECT SUMMARY

Overview:

This proposal requests support for improving the performance, scalability and reliability of Parallel Discrete Event Simulation (PDES) systems on emerging multi-core platforms. Discrete Event Simulation (DES) is used pervasively for performance evaluation across many disciplines including computer and networking system design. With the transition of commodity microprocessor platforms to multicores and manycores well on its way, there is an increasing need to exploit parallelism as the primary vehicle to improve performance and capacity of applications. To improve the performance and scalability of PDES, research into alternative simulation algorithms and data structures that are able to maximize parallelism, and improve load balancing, while maintaining locality is proposed. A theme of many of the solutions is to use model analysis to derive exploitable characteristics from the model, and use them to improve the run-time. The research also explores how manycore architectures could be designed to best support fine-grained applications such as PDES. Research to improve the reliability of PDES is also proposed: the performance of PDES is shown first to be extremely sensitive to interference from the system and other colocated workloads. The problem is explained and acceptable slowdown for resilience is defined. To reach idealized slowdown, a number of techniques both at the application and the system level are proposed. Finally, to protect against hard failures, techniques to exploit simulation time to develop lightweight check-pointing to enable recovery from failures are proposed.

Intellectual Merit :

The proposed work contributes in a number of ways to knowledge advancement in scalable high performance computing: (1) Developing algorithms and data structures to improve performance and scalability
of PDES in manycores and clusters of many cores; (2) Exploiting model information to improve simulation performance using static analysis combined with run-time dynamic adaptation; (3) Using PDES as a benchmark to explore and guide architecture design decisions and specialized hardware elements to better support this type of application; and (4) Developing algorithms and models for improving resilience to performance anomalies and failures.

Broader Impacts :

Improving PDES performance translates to improved simulation (more detailed models, shorter evaluation cycles) in a wide range of application domains. The lessons learned from this project can
be beneficial in improving the performance of fine-grained applications in general. In addition, PDES represents a detailed benchmark that can inform decisions about manycore architectures, especially in the area of inter-chip communication and the memory hierarchy integration. Finally, the project will enable the PIs to develop courses in PDES and manycore system programming and to train students in this emerging area.

1337213

TABLE OF CONTENTS

For font size and page formatting specifications, see GPG section II.B.2.

*Proposers may select any numbering mechanism for the proposal. The entire proposal however, must be paginated. Complete both columns only if the proposal is numbered consecutively.

TABLE OF CONTENTS

For font size and page formatting specifications, see GPG section II.B.2.

	Total No. of Pages	Page No.* (Optional)*
Cover Sheet for Proposal to the National Science Foundation		
Project Summary (not to exceed 1 page)		
Table of Contents	1	
Project Description (Including Results from Prior NSF Support) (not to exceed 15 pages) **(Exceed only if allowed by a specific program announcement/solicitation or if approved in advance by the appropriate NSF Assistant Director or designee)**	0	
References Cited		
Biographical Sketches (Not to exceed 2 pages each)	2	
Budget (Plus up to 3 pages of budget justification)	7	
Current and Pending Support	2	
Facilities, Equipment and Other Resources	1	
Special Information/Supplementary Documents (Data Management Plan, Mentoring Plan and Other Supplementary Documents)	2	
Appendix (List below.) **(Include only if allowed by a specific program announcement/ solicitation or if approved in advance by the appropriate NSF Assistant Director or designee)**		
Appendix Items:		

*Proposers may select any numbering mechanism for the proposal. The entire proposal however, must be paginated. Complete both columns only if the proposal is numbered consecutively.

1 Project Overview

This proposal explores supporting scalable, high performance, and resilient Parallel Discrete Event Simulation (PDES) on emerging multicore and manycore clusters, and in turn, how the architecture of these systems can evolve to best support fine-grained applications such as PDES. Discrete Event Simulation (DES) is used widely for performance evaluation during the design and analysis of systems where changes of state are discrete [35, 55, 106]. DES is used across a wide range of application domains including in the simulation of computing and telecommunication systems, automotive traffic networks, operations research, biological systems, war gaming, training simulations and many others. Many of the principles of DES also underlie distributed virtual reality and gaming systems [40]. As the scale and complexity of the systems being modeled continue to increase, PDES offers the promise of improving the performance and capacity of DES, allowing the simulation of larger, more detailed, models in faster times. Improving the performance and scalability of PDES [36, 40] can have substantial impact on many disciplines.

1.1 PDES and Multicores/Manycores

The arrival of multicore processors, and their ongoing evolution into *manycores*, makes parallelism the primary vehicle for improving application performance [2, 41]. Manycores provide an environment with ultra-low communication latencies that can substantially impact fine-grained application such as PDES whose performance and scalability are often limited by the communication cost. Traditionally, studies with PDES algorithms on Beowulf Clusters are almost singularly focused on tolerating the impact of latency because of its dominant effect on performance [18]. Relatively less attention is paid to other aspects of the simulator because the impact of other optimizations is secondary relative to communication [5]. As on-chip latency is significantly reduced on manycores, performance bottlenecks shift to more typical algorithmic, load-balancing and synchronization problems. In addition, the deep memory integration available on manycores enables new optimization opportunities.

At the same time, manycore architectures are evolving quickly and computer architects are seeking feedback from applications to guide their design. There have been few quantitative investigations of important classes of applications and the impact of architectural design decisions on them [44, 100, 54, 64, 53, 11, 71, 46, 49]; however, deeper investigations rooted in important applications are needed. Design constraints such as the high on-chip power consumption forecast that most of the silicon on future CPUs will be dark [29]. This trend invites investment in custom accelerators that would be powered up only when needed, rather than having homogeneous cores that cannot be all active. While generic accelerators and frameworks to support them have been proposed, we believe that custom accelerators for important classes of applications also hold promise.

Finally, PDES performance (and in fact, that of many parallel applications) suffers disproportionately in the presence of interference from co-located applications and/or system services. The slowdown experienced generally far exceeds expectations; for example, in one of the simulators we studied, the presence of even a single interfering process caused unbounded slowdown of an 8-way simulation. The problem occurs because in an application with dependencies, when one thread is inactive due to a context switch, the remaining threads may not be able to proceed while they wait for dependencies to resolve. This problem is known in the parallel processing community and techniques such as gang-scheduling are used to ensure that all the processes execute together with none of them experiencing context switches [32]. While gang-scheduling is supported by some resource management utilities, it is not supported in many installations. Gang-scheduling can also be inefficient in the presence of I/O [103]. Thus, we believe that it is essential to design simulators (and parallel applications in general) in a way that is tolerant to interference and that can approach idealized

graceful slowdown. Furthermore, hard failures are likely to become increasingly common at large scales and for long-running simulations. We believe that *enabling resilient execution must be a first order design requirement for truly scalable applications*.

1.2 Proposal Objectives

The overall goal of the proposal is to *enable high performance, scalable, and resilient PDES on emerging manycores and clusters of manycores*, by addressing the challenges described above. In particular, the specific research objectives are:

1. **Building a Simulator for Manycores and Clusters of Manycores:** In our current effort, we built a multi-threaded simulator primarily by rewriting the communication library of a process-based simulator. As a result, the algorithms and data structures used in the simulator are not optimized for multi-threaded operation, especially as the number of threads scale. The current simulator also does not exploit the opportunities available from the shared address space and tight memory integration of manycore environments.
2. **Exploiting Model specific Information in the Simulation:** Building a general simulation engine for all models promotes ease of development and parallelization of simulation models. However, there is significant opportunity for accelerating simulation based on the knowledge of the model behavior; this opportunity is not typically exploited in general simulation engines (and especially not by the parallel simulation community). We will pursue using analysis of the simulation model to identify exploitable information in the model and opportunities to use this information to optimize run-time performance.
3. **Exploring architecture and system support for fine-grained applications such as PDES:** Tremendous improvements in superscalar architectures occurred only after architects started basing design decisions on quantitative analysis of performance on typical programs [52]. In contrast, manycores are evolving without a corresponding deep understanding of the processing characteristics of parallel applications. We will investigate using PDES to explore manycore architecture space in the following ways. We will evaluate how the manycore organization, including features such as the number and nature of cores, cache sizing and organization, on-chip network, and others should be designed to best serve PDES. In addition, we will explore the use of custom hardware support and accelerators to improve the performance of PDES.
4. **Developing Performance-resilient Failure-resilient PDES algorithms:** We propose to analyze the performance resilience problem to explain the reasons behind the problem. We also propose PDES organizations that allow remapping of work at different granularities to provide resilience to these performance anomalies. To provide efficient failure resilience, we propose light-checkpointing and recovery algorithms that exploit the nature of simulation (simulation time) and the checkpointing/Global Virtual Time (GVT) algorithms present in optimistically synchronized simulation [40].

In addition to the research objectives, one of our goals is to contribute to the large need for trained professionals who are able to effectively program manycores. We will pursue this objective by training students and developing pedagogical material; especially focusing the training and materials for achieving performance-resilient and failure-resilient parallel computing.

1.3 Broader Impacts

Together, the research objectives improve the performance, scalability and reliability of PDES: an important application that can impact a wide range of areas that use it. In addition, PDES is a representative of

an important class of applications: we believe that many of the techniques we propose can be adopted or modified for other applications. The use PDES to analyze manycore architectures and to investigate hardware support for fine grained applications provides the architecture community of an important large scale benchmark that is challenging to parallelize to help guide future manycore designs. The techniques developed to improve performance and failure resilience can be applied to other parallel applications; such techniques are integral to the design of scalable applications. All simulators, benchmarks, and tools will be released to the public.

Finally, the project will lead to the training of students, both those that work on the projects, and those benefiting from a new course being designed, in an area where there is an urgent need at a national level for trained experienced professionals. The PIs have a track record of involving women and underrepresented minorities in research as well as working with undergraduate students.

1.4 Prior Support and Research Team

The proposed research builds on an existing collaborative NSF grant (CNS–0916323 and CNS–0915337) which is currently in its final year. The grant explored building a simulator and characterizing its performance on manycore environments. In [5] we analyzed the performance of two existing simulators on a multicore cluster and identified some of the major bottlenecks. We developed a multi-threaded version of both simulators, and carried out basic optimizations to improve its performance [47]. We evaluated the use of Dynamic Voltage and Frequency Scaling (DVFS) in improving performance and energy efficiency of PDES [21, 22]. We also pursued an example of exploiting model characteristics in improving simulation performance [4]: in particular, we analyzed the model to guide partitioning of the simulation based on dynamic behavior, rather than static connectivity information. We also explored optimizing simulator on manycores such as the Tilera Tile64 [48] and the Intel SCC [21].

This prior work frames the context for the current proposal. We have developed the base simulator for manycores and implemented a number of optimizations to it for a number of different multicore/manycore organizations. We also have preliminary evidence that exploiting model information can significantly improve performance; up to 4X improvement in performance resulted when we exploited model information in partitioning [4]. The proposed work builds on both the insight gained from these studies and the simulator and optimizations to continue improving performance, scalability and reliability of PDES.

The project team has extensive and complementary expertise in PDES (Dr. Wilsey and Dr. Abu-Ghazaleh), including system level optimization, including both static analysis and dynamic adaptation. The PIs also have expertise in multicore/manycore architectures (Dr. Ponomarev), as well as general parallel processing (all three). Although the research agenda we outline is aggressive, we believe that with two graduate students for four years, and a third for three years, we have a realistic chance of successfully executing it.

2 Context and Background: Parallel Discrete Event Simulation

In DES, a model consists of a set of simulation objects with associated state [35, 55, 106]. 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 set 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 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 [40]. 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 [6, 36, 70, 88]. Correct simulation requires that all events be processed in their causal order (or, more precisely, that the PDES execution produces simulation results that are consistent with the DES execution results; generally this is achieved by preserving causal order). Therefore, a synchronization model is needed to ensure that remote events are processed in their proper causal order. There are two main synchronization protocols for parallel simulation, namely: *conservative* and *optimistic* [36, 40].

Conservatively synchronized PDES simulators are designed to process events so that the causal orders between events are always strictly satisfied. Each LP must receive assurances from others (or know from static analysis) that it may proceed, severely limiting the potential processing concurrency. In contrast, optimistically synchronized PDES simulators following the Time Warp protocol do not enforce causality during event processing. Instead, checkpointing is used to enable recovery from causal violations (which are detected when an event is received with an earlier processing time than the current simulation time) [50]. Such an event (called a *straggler event*) is recovered from by rolling back the simulation to an earlier state and canceling out 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, uncontrolled 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 [40]. To control these rollbacks, a technique called throttling is often used [1, 7, 26, 34, 33, 76, 94], where the speculation is limited to a certain number of events, or a certain simulation time beyond Global Virtual Time (GVT). With throttling, we can restrain overly optimistic processing. However, throttling is a coarse grained approach that does not take into account the nature of the individual dependencies.

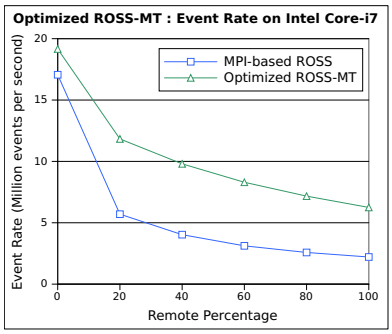


Figure 1: Event Processing Rate (Intel core-i7)

3 Proposed Research

In this section, we present the proposed work organized into the four primary objectives of the proposal: (1) Redesigning PDES for manycores and clusters of manycores; (2) Exploiting application information to improve performance and scalability; (3) Exploring architecture design and hardware support for PDES; and (4) Improving the resilience and reliability of PDES.

PDES exhibits limited scalability due to its fine-grained nature and dynamic dependencies: On clusters, it often results in reverse speedup (*i.e.*, slowdown) beyond a certain number of processors, as the impact of communication latency outweighs the benefits of additional parallelism [5]. However, on systems where the cost of communication is low, significantly better scalability is observed [78, 8]. In earlier work, we

developed a multi-threaded implementation of a parallel simulator that enables direct communication between threads running on the same machine [47]. To support the multi-threaded implementation, locks were placed around shared data structures; these locks often caused performance bottlenecks, especially on systems where the number of threads is high. As a result, we implemented a number of optimizations to improve lock contention, to localize access in the presence of Non-Uniform Memory Access delays (NUMA) and to optimize group synchronization operations [47]. As can be seen in Figure 1, the results on the core i7 running an 8-way simulation of the Phold Benchmark [36] show that the multi-threaded simulator more than doubles the event processing rate of the baseline MPI based simulator. However, it only outperformed the MPI version by 25% on the 48-core AMD Magny-cours machine due to the inefficiency of the synchronization costs under high contention.

3.1 Objective 1: Redesigning PDES for manycores

In essence, our multi-threaded implementation consisted of a complete rewrite of the communication primitives, but left the core of the simulator relatively unchanged, implementing a few optimizations to improve a design that was not targeted for concurrent operation. Thus, our first goal is to explore designing a PDES simulator for manycores.

Under this objective, we anticipate the following activities:

Multithreaded Simulator Design: Due to the high impact of communication latency, traditional PDES systems targeting Beowulf clusters are focused on hiding the impact of latency; the many other algorithms and data structures for PDES have a small impact on performance. However, on multicore clusters, the low latency moves the bottlenecks to other aspects of simulator operation (*e.g.*, buffer management [37], cancellation and rollbacks [38], and so on). Moreover, the tight memory integration on a multicore chip opens the door for new optimizations that take advantage of the ability to more closely monitor the progress of other threads and redistribute the work among them if necessary. In addition, the low communication latency can make the progress rate of the different simulators more predictable, reducing the need for techniques that are designed to tolerate very high latency in favor of more synchronous and predictable simulation cycles.

Synchronization-Friendly Data Structures: As the number of cores in manycore processors grow, it becomes increasingly challenging for application software to take advantage of the additional parallel processing capabilities. The principle complications come from contention for shared data and the slowdowns caused by the hardware memory coherency interlocks [41]. From the standpoint of PDES, one of the key challenge areas lies in the need for solutions providing effective, contention-free data structures along the critical path of the simulator (event processing). This is especially true since many discrete event simulation models tend to have relatively fine-grained computational requirements making frequent access to these data structures necessary.

To show an example of the design of synchronization-friendly data structures, we propose a software architecture for the critical event management list that is manycore friendly. The event management list holds the pending events sorted by their simulation time to enable the Least Time-Stamp First (LTSF) scheduling needed by the simulator. In particular, we propose a design that uses a modified ladder queue [93] data structure to hold the pending event sets. While splay trees, calendar queues [14], or lazy queues [90] are often considered the data structures of choice for managing the pending event list, ladder queues [93] offer some interesting properties for multi-threaded simulators. The key advantage of the ladder queue is that the structure is relatively independent on the time granularity of the simulation model. Ideally, the range of event timestamps for the buckets that define each epoch of the ladder should help ensure that the events within any bucket of the ladder queue are causally independent. This idea is related to the lookahead property of conservatively synchronized parallel simulation [36]. The lookahead concept results from a static

analysis of the simulation model that guarantees a time window ahead of any source timestamp in which no additional events will be generated (effectively a minimum time delay guarantee on the event processing latency).

While not as strong as lookahead, a key hypothesis for this proposed structure is that *the bucket size for the ladder queue will typically encompass a range of event timestamps that are causally independent*. Thus, we propose to explore a modified version of the ladder queue that removes all sorting and relies on a weak causal relation of the events within a single bucket to significantly improve the performance of this critical data structure. When the sorting requirement is removed, it should be possible to use atomic moves to manage the buckets, resulting in a fully wait-free, contention minimal pending event list data structure. Note that if two events happen to be causally dependent in the same bucket, simulation correctness is preserved by the rollback that occurs when the later event is encountered for an earlier local time. Of course this solution may not work well for simulation models with tight causal relations between adjacent (or nearby) events. That said, this failure will occur only for simulation models where the Time Warp mechanism itself would also not generally be effective anyway and thus, no loss of generality in the use of a Time Warp synchronized parallel simulation solution occurs.

Similarly, we plan to explore alternative designs for other critical data structures, including those used for event communication, state saving, event cancellation, and so on.

Supporting Clusters of manycores: Scalability requires supporting simulations across large clusters of manycores. The work discussed thus far considers only a single multicore/manycore machine. In a cluster of multicores/manycores, the delays between cores on different nodes can far exceed those between cores on the same machine. Moreover, typically, cores on different machines do not share the same address space, requiring expensive message passing operations across the network. As a result, we observe that simulation performance suffers substantially in the presence of these expensive communication links.

We propose to address this problem in the following ways. First, we propose to improve partitioning and object migration algorithms to reduce the frequency of communication across the slow links by exposing the high cost of these links to the partitioning algorithm. Second, we propose to use message combination to reduce the overhead of message communication. In particular, since there are many threads on each manycore node, we are able to aggregate messages from any machine to any other machine regardless of the source and destination threads on those machines. This aggregation in space is substantially different than classical message consolidation which aggregates messages that share the same source and destination where we have to delay communication in hope of receiving additional messages for consolidation [85]. Finally, we propose to use two tiers of synchronization where cores on the same machine are tightly synchronized while loose synchronization is supported across machines.

Concurrent dynamic memory management support: Once the communication bottleneck is alleviated, one of the potential new bottlenecks, the memory system bandwidth, will start to play a more pronounced role [72, 73]. The cores on a chip share a single system bus to main memory. It is important to note that this is a bandwidth problem [15], rather than a latency problem as per the classical memory wall [104]; there is hope that this problem can be reduced with improved integration of processing and memory [69, 68, 30]. For most models, we do not expect PDES to exhibit data intensive properties. However, memory pressure will increase as the number of cores increases. Moreover, in cases where the models have large memory working sets, or when simulations across multiple chips are used, the simulation can become limited by the off-chip bandwidth to the memory. For this reason, it is important to develop solutions that reduce the pressure on the memory subsystem. To this end, we will develop support for concurrent dynamic memory management in user space to reuse and share the available memory to avoid unnecessary memory traffic and to avoid false sharing. The memory manager will also be aware of NUMA characteristics of the underlying

system and will attempt to match requests to memory regions close to the requesting thread.

The anticipated outcome of this objective is a highly efficient multi-threaded version of the simulator, with advanced synchronization friendly data structures. In addition, the simulator should scale to clusters of manycores by using efficient partitioning, message consolidation and other techniques to optimize communication across the expensive networking links.

3.2 Exploiting Model Information

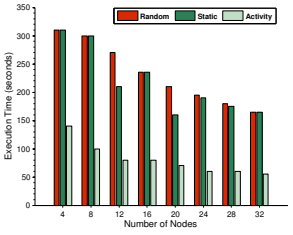
The simulation kernel approach that we use in this proposal promotes ease of development of parallel simulation models. The domain experts develop their models with little or no awareness of the fact that it executes in parallel. In contrast, custom parallel simulators are often created for specific domains or even models. This approach requires advanced expertise and large development times. However, such simulators are able to exploit the nature of the model or the domain to achieve substantial efficiencies. In contrast, the simulation engine approach typically does not exploit these opportunities. The goal of this component of the research is to exploit model information to improve the performance and scalability of PDES.

Static analysis refers to compile time analysis and optimization of the simulation model. Although static analysis holds great potential for accelerating PDES, playing a role much like that of a compiler in optimizing source code and providing hints to the run-time system, it has not received significant attention by the PDES research community. We do not seek within the scope of this effort to develop a full static analysis tool (the equivalent of an optimizing compiler for the modeling language, which supports concurrency). Instead, we focus on a small number of *dependency related analyses* that attempt to improve the time management (synchronization and concurrency) and load balancing aspects of the simulation. These techniques do not transform the model at all and are orthogonal to techniques that do so; rather, they focus on understanding the structure of the model to provide insight into how the simulation kernel should execute it.

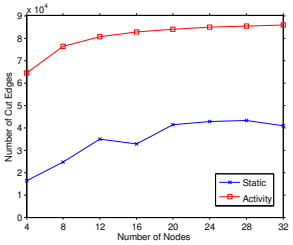
Static analysis will work in conjunction with the dynamic techniques, both implicitly by providing an effective starting state, and explicitly by providing run-time hints, to maintain the simulation in a highly efficient state. By focusing on the dependencies, we avoid complex optimizations that require a full-featured modeling language; in fact, the techniques should be independent of the modeling language and domain.

Example–Exploiting model information in partitioning: One tool for reasoning about model dependencies is the static model connectivity graph. In this graph, vertices represent objects, and edges represent dependencies between objects. An object communicates only to objects it is directly connected to. Partitioning tools derive this graph and partition the model based on it into roughly equal size pieces (typically measured in terms of number of objects) while minimizing the cut size using classical recursive bi-partitioning or multi-way partitioning (*e.g.*, [57]).

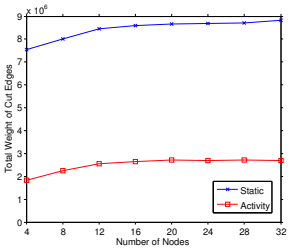
While this partitioning approach improves locality and load balancing over arbitrary object placement, using the static connectivity graph does not consider the relative importance of the edges within the model. In particular, it is common in models to see large variations (even fat-tailed distributions) of the relative activity of edges in the communication graph [56, 31]. Consider, for example,



(a) Execution time



(b) Static Mincut



(c) Active Mincut

Figure 2: Importance of Exploiting Model Information

the importance of links on the Internet topology: a link at the core of the Internet carries orders of magnitude more packets than a link near the edge [56].

In recent work, we exploited model information in partitioning the model for parallel execution [4]. In particular, we examine the model to estimate the importance of the communication links between the objects. By partitioning in this way, we are able to substantially improve performance for some models (see Figure 2(a)). This improvement happens because rather than optimizing the Static mincut, as traditional partitioning tools are able to do, we are able to match the partitioning to the application behavior, minimizing the active mincut instead (the number of communication messages and their criticality as estimated from the model), resulting in large improvements in run-time.

Towards the objective of exploiting model information to improve performance and scalability, we propose the following efforts.

Extracting Dependency Patterns: Towards Data flow Simulation: The goal of this research is to use the causality information between events to determine when computation can proceed optimistically and when it should be throttled because of an expected incoming event. If we accomplish this goal, we end up with a data flow simulation where LPs wait only when necessary to avoid a rollback. Contrast this model with conservative simulation, where LPs have to wait even when it is unnecessary, or optimistic simulation, where LPs never wait but require expensive checkpointing and often miss critical dependencies resulting in cascading rollbacks and computational thrashing. To be able to infer when a dependent event can be expected, we need to derive the causality relationships of events to each other to extract patterns of event arrivals. Of course, this is not always possible using static analysis; therefore, additional dependencies can be tracked at run-time. While we expect the range of event patterns that can be exploited to be large, we discuss below examples of these patterns.

1. **Predictable event patterns:** When an object generates events in a predictable pattern, this pattern can be passed on to directly connected objects. These incident objects, knowing the expected arrival pattern, can then wait for the event when its expected arrival time is reached in the simulation. The pattern can be periodic or driven by the state of the system, in which case we should identify trigger events that cause the patterns to start and use them to initiate/stop throttling. Throttling can be implemented using a soft lookahead; that is, wait for an expected message for a period, but progress if it does not arrive to avoid deadlock and the need to send null messages.
2. **Triggered Cascade patterns:** Consider a sequence of events that are causally dependent. For example, event `a` causes event `b`, which in turn causes event `c`. We note that this cascade pattern is not constrained to a linear chain, rather any propagating set of events can be targeted. If these events are critical, the object generating the initial event may inform the objects in the cascade of the generation of the event so that they wait for the event to propagate to them before proceeding past its simulation time. In the event of a rollback at object generating the event `a`, all of the cascade objects may be informed directly by the object that was rolled back to limit propagating rollbacks (the standard implementation has to wait for the rollback event to cascade down the object chain).

We note that for criticality to be relevant to the run-time behavior of the events, the progress time of the different LPs must be uniform, or known (to adjust the criticality of the event with the virtual time difference between the processes). It is likely that the simulation will exhibit more uniform progress due to the low degree of optimism needed to hide latency, making it possible to estimate criticality. The run-time support will target maintaining this uniform progress rate across the LPs, by continuously detecting load imbalances and migrating objects if necessary to maintain the simulation in a load balanced state.

Assisting with configuration information: PDES behavior is influenced by algorithmic choices and configuration parameters for the various subsystems within it. Choices such as the use of aggressive or lazy cancellation [19, 59, 84, 87], optimism control [1, 26, 33], incremental or periodic check pointing [60, 75, 89], GVT computation[9, 10, 27, 39, 66, 95], event scheduling algorithm [61, 80, 92] and data structure [14, 23, 24], communication alternatives [18, 85], and others can profoundly influence the behavior of the simulation [26, 43, 82, 83]. We expect static analysis to assist in selecting the simulation configuration that best matches the model as described by the dynamic dependency graph. We note that the choices need not be applied uniformly across the model; different objects may choose different alternatives. For example, for a critical event, we may choose a cancellation policy differently from that used for a non-critical event that will not cause cascading rollbacks. We may checkpoint an active process more frequently than an inactive one. Static analysis may also offer hints about how this configuration should be adapted as the simulation evolves over time.

3.3 Dynamic Event-level Run Time Monitoring and Adaptation

The simulation behavior in PDES can be highly dynamic, often in a way that resists static analysis because of issues of scale as well as data-driven, or dynamically changing dependencies. Additionally, the static analysis directives assume uniform or nearly uniform progress of the different LPs such that the model event criticality corresponds to the run-time criticality for the different events. Dynamic run-time monitoring has the benefit of observing the behavior of the simulation in real-time, rather than having to derive it. At the same time, the available time and the overhead of run-time adaptation necessitate the use of lightweight distributed techniques that are applied judiciously based on partial information [82]; we cannot afford the sophisticated analysis that is possible at compile time. Thus, static analysis and dynamic adaptation can work in concert to bring their complementary strengths to bear on the problem of directing the PDES simulation into an efficient state.

The run-time system must act on and complement the hints provided from the static analysis. Static analysis based hints may be incomplete, inaccurate, or predicated on observed dynamic behavior. The primary roles we envision for the dynamic run-time monitoring and adaptation are as follows:

1. **Dynamic dependency tracking:** The run time system can selectively track the criticality and activity of the different dependencies and objects. For example, we can restrict this tracking for high activity dependencies or those dependencies that are causing rollbacks. This information can then be used to create lightweight versions of the coordination techniques that were proposed under static analysis to cover dependencies that static analysis was not able to characterize and to evolve the estimates as the simulation behavior changes.
2. **Dynamic simulation control:** The dynamic run-time system must track the changing behavior of the simulation and respond to it to reconfigure the simulation *on-the-fly* by adjusting the simulation algorithmic choices and their parameters (which could be different for different groups of objects) [82]. The impact of the simulation configuration parameters in a manycore environment must be carefully evaluated to inform the adaptation triggers and strategies used. Again, we hope to be able to make some of these decisions based on static analysis, and then use the run-time monitoring to complement/expand these decisions at run time.
3. **Dynamic object migration:** The goal of dynamic object migration is to repartition the simulation as it evolves in time, resulting in a different dynamic dependency graphs than the one present at the start of the simulation. Object migration can be then used to localize critical dependencies and to load balance the computation across the different cores. Moreover, one must be careful that event messages

in progress find their way to the destination LP. We intend to support object migration by creating copies of selected objects (those that are sufficiently active to warrant consideration for migration). Multiple implementations are possible: in one implementation all but the primary receive messages but do not send messages. Migration then involves changing the active copy. If a rollback occurs to an earlier time than when the responsibility of the current copy started, the rollback is forwarded back to the previous copy of the object to restore the state (thus avoiding the need to serialize and transfer the state and event histories, which in most cases would simply be garbage collected as GVT progresses beyond them).

4. **Maintaining a uniform progress rate of the simulation LPs:** Given the more predictable nature of simulation progress due to the lower communication latencies and more aggressive synchronization, the primary issue in maintaining uniform progress is load balancing. Since for some models, object activity may change over time, maintaining load balancing using object migration only may not suffice. Thus, we will explore the use of target progress rates that can be used to prevent LPs from moving far ahead of other LPs in low activity periods.

3.4 Objective 3: Architecture Support for PDES

In experimenting with our multi-threaded simulator on three different multicore platforms (Intel corei7, AMD Magny-cours [47], and the Tilera Tile64 [48]), the simulator exhibited substantially different performance and scalability patterns. The platforms differ significantly in the core organization, the connectivity between the cores, and the memory system properties. The goal of this objective is to understand how the architectural design choices influence the performance of PDES, and explore alternative architecture organization and support to improve the performance and scalability of PDES.

Towards this goal, we have already successfully executed the multi-threaded PDES on MarSSx86 — a cycle-accurate multicore processor simulator [77] which was developed at SUNY Binghamton with the support of a previous NSF grant. Thus, we are able to study the interaction of the simulator with the underlying architecture and understand the impact of architecture choices on the performance of the simulation. Using this environment, we propose to the following research studies.

Many core architecture design space exploration: we propose to study the impact of the different design choices in multicores on the performance of PDES. We hope to be able to answer questions such as: how should caches be sized for PDES? Is it better to have communication through cache sharing or through a dedicated on chip network, or both? Is it better to have more simpler cores, fewer more powerful cores, or a mixture of both? Can we model current manycore proposals and evaluate them? Can we identify bottlenecks in different architectures and use that information to influence the PDES algorithm design?

Answering these questions requires substantial improvements to the MarSS simulator. It is currently limited to running a 16 core machine and is quite slow. Although it models multicores it does so using shared caching, but does not have models for chip interconnect or tiled caches. Although we are interested in cycle accurate simulation of the memory and the communication systems, behavioral models of the CPU are likely to be sufficient. Parallelizing the MarSS simulator is also a possibility to improve performance.

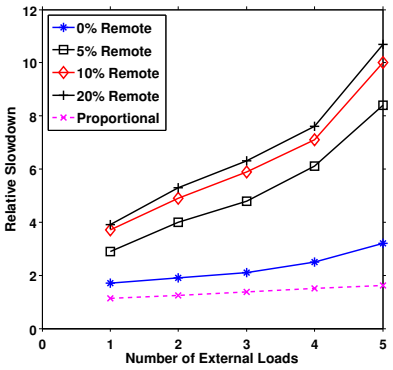
Exploring architecture support for PDES: in some estimates, the power wall will prevent future CPUs from having a large number of cores that are always active. Instead, most of the silicon has to be dark [29]. One model to use the chip real estate is to build dedicated co-processors that are of use for important applications and are thus turned on only when these applications are active. In this context, we will explore dedicated architecture support for critical components of the simulation system. Candidates include hardware synchronization primitives [65, 74], support for communication and checkpointing managers, and

hardware support for critical data structures [63]. These architecture features will be modeled in MarSS and if they prove beneficial, they will be prototyped to accurately estimate timing, size, and power impact.

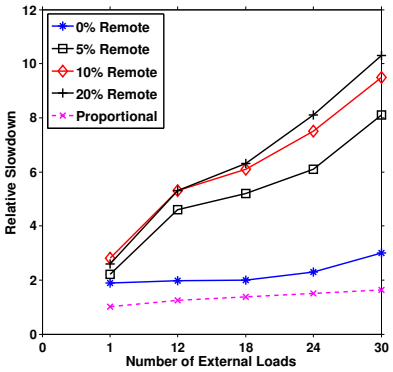
3.5 Objective 4: Performance and Failure Resilience

Performance-resilience and failure-resilience are key properties for scalability. Performance resilience is the ability of the simulator to tolerate variations in the operating environment, application behavior, and/or their interaction with acceptable impact on performance. On the other hand, as the scale of applications increases, requiring larger systems, and execution for longer times, the probability of system failure increases. Thus, for scalability, the application must be able to tolerate failures and complete its execution. Our goal is to improve the performance and failure resilience of PDES.

Improving Performance Resilience: Parallel applications, including PDES simulators, are often designed under the assumption of a homogeneous environment with no interference from other co-located applications or the system. Such interference creates competition for the available resources leading to potential slowdowns. The slowdowns far exceed those predicted by the proportional loss of resources due to interactions with the OS scheduler. In particular, we define a metric, *proportional slowdown*, that predicts the performance based on the loss of resources due to the interfering load. For convenience, let us assume that the critical resource is CPU cycles. Thus, in the presence of a single interfering process we expect the relative runtime on an N processor machine where the simulator is running on all cores to increase to $\frac{N+1}{N}$. However, as can be seen in Figure 3, the effect is significantly higher than that.



(a) Intel Core i7 System



(b) AMD Magny-Cours System

Figure 3: Relative Slowdown of ROSS-MT caused by External Loads

The problem occurs because the operating system scheduler context switches one of the application threads which now becomes inactive. Due to the dependencies in the application, the loss of performance is not only the time that this thread loses. The remaining threads cannot proceed until this thread returns to activity; the situation is even worse for optimistic simulation where the running threads proceed optimistically and have to be rolled back once the context switched thread returns. Depending on the scheduler and the density of the dependencies, the performance of the simulation can be dramatically impacted – on some machines, the simulation fails to make progress in the presence of even one interfering process.

This problem is known in the parallel processing community, and has led to the development of techniques such as gang-scheduling to prevent threads from a parallel application from being context switched [32]. Gang scheduling support is being built into Linux; however, it must be supported across the whole cluster, not just by the individual nodes. Advanced resource managers and schedulers for cluster environments sometimes support Gang scheduling or similar advanced schedulers [105]. However, Gang scheduling can lead to loss of efficiency and poor sharing properties under some conditions (e.g., in I/O bound phases of the application) [103]. Moreover, sometimes the resource scheduler use is advisory, and interfering loads can be created by users that do not use the scheduler. Thus, we believe that building resilience in the application can alleviate these problems, and even provide better properties than

Gang scheduling.

Dynamic object migration [102, 86] and similar approaches for load balancing [96] do not help with this problem unless the work is completely migrated from the blocked thread. At the same time, work stealing is difficult from the context-switched thread because it may be interrupted at an arbitrary point in processing while holding locks.

We propose the following approach to address this problem. First, we need to investigate how to reliably detect the onset of the problem using approaches such as having each LP monitor the simulation time of other LPs to see if they are trailing significantly. Once a performance anomaly is detected, there are a number of possible reactions. We may disable one of the processing threads and remap the work (temporarily or permanently) to the other threads. We must do this in a locality aware way so that each hardware thread remains primarily associated with its set of objects, but may assist periodically with the events for the lagging thread.

Finally, to help with the problem of work stealing from a context switched thread, we will explore a number of approaches. The work stealing scheduler implemented within the Cilk run-time environment is one promising approach [12]. We will also explore changing the Operating System scheduler to make it more friendly to applications. In particular, we propose the idea of polite context switches where the OS informs the process to be context switched of an imminent context switch allowing it to reach a safe state with respect to work stealing before it is interrupted. The process can then voluntarily yield the CPU, or after a short period of time receive the usual timer interrupt. Informing the application of the state of the scheduler promotes effective reaction to the system state and better resilience to interference. For example, it may be used as the trigger for deactivating a thread rather than having to detect a performance anomaly and react to it.

Failure Resilience: we also propose to support failure resilience to enhance application scalability as failures become more common at larger scales in number of nodes or run-time. We leverage the presence of simulation time to implement lightweight checkpointing. In particular, in optimistic simulation, computing the Global Virtual Time problem has been shown to be very similar to the problem of consistent distributed snapshots [17, 66]. Thus, with small extensions to GVT, the checkpoints being taken by the simulation to support rollbacks can be re-used to provide failure protection. What remains is to implement a low overhead background mechanism to distribute snapshots to multiple machines to provide redundancy in the availability of the checkpoints.

4 Experimental Approach and Alternatives

We will use the ROSS [16] and WARPED [81] simulators in our experiments. We have significant experience in working with both environments, and together they provide a range of algorithmic choices, optimizations and simulation models that will allow us to carry out high quality experimental evaluations.

We recognize that performance evaluation represents a difficult challenge for projects such as ours that target an emerging architecture class. We address this problem in two ways. First, we have access to a number of interesting manycore nodes, and a state of the art cluster of multicores (32 machines, each with 64-cores) that will allow us to do experimental evaluation. In exploring the architecture design space, we will be working with the MarSS simulator which will allow us to model the proposed architectural features and evaluate them. MarSS is a full system simulator/emulator that includes models of conventional multicores. We will extend MarSS to support features such as on-chip communication networks and tiled architectures representative of emerging manycores. We will also consider using RAMP (an FPGA based emulator for future multicore systems [99]) and its associated simulator RAMP Gold to have access to a wider range of

multicore design points.

5 Related Work

In this section, we discuss the prior efforts most relevant to the current proposal, including PDES algorithms for shared-memory multiprocessors, static analysis, as well as dynamic adaptation of PDES.

PDES Optimizations for Shared Memory Multiprocessors: While multicore and manycore systems are a relatively recent phenomenon, shared memory multiprocessors (where the individual processing elements are located on separate chips but communicate through shared memory) have been extensively researched. While some studies with conservatively synchronized PDES on shared memory multiprocessors also have been performed [3, 62], we will focus on the optimistic studies that are most closely related to the planned studies in this project. Fujimoto *et al* examined the design of optimistic PDES systems and algorithms in the context of cache-coherent shared memory multiprocessors [25, 39, 37]. These works proposed a series of new algorithms, including ones for buffer management, message cancellation and GVT calculation, specifically targeted for shared memory environments. For example, in shared memory system, the state is shared, allowing an LP to cancel out erroneously sent remote events directly, eliminating the need for anti-messages [25]. Similarly, in shared memory, messages can simply be written into a buffer and become visible to all processors. The exploitation of such features allows the GVT computation algorithm to be implemented through a single round of inter-processor communication (as opposed to at least two rounds required for message passing programming model) with minimal number of shared variables and data structures. Fujimoto and Hybinette also describe an efficient *on-the-fly* fossil collection algorithm to enable fast reclamation of memory [39]. They also explore efficient buffer management algorithms for shared memory environments [37].

While some of the schemes developed for shared memory machines can apply for manycores, the tighter coupling of processing elements and shorter communication delays for accessing shared caches requires at least a careful reconsideration of these approaches. Furthermore, previous PDES designs for shared memory used simulations with high degree of optimism, which may no longer be necessary with short communication latencies.

Exploiting Application Behavior in PDES: Perhaps the most widely known and successful example of static analysis is levelized code compilation (LCC) for logic simulation [98]. In this approach, the gates in a simulated sequential circuit are organized in terms of depth and compiled into code that evaluates all the gates breadth first starting from the inputs in every cycle regardless of the presence of events. This native execution obviates the need for event driven simulation yielding dramatic improvement in performance. Beyond LCC, there exist few examples of static analysis: static model partitioning [57, 58] can be considered a type of static analysis. Our work improves on these approaches by considering the dynamic dependency graph rather than the static object connectivity. McBrayer and Wilsey's object combining develops the mechanism for combining objects (in their case, processes in VHDL models) to reduce the number of events and increase the granularity of processing [67]. Our approach should be able to systematically identify effective objects for combination based on event criticality and footprint. However, this represents an instance of model transformation that we will pursue only as time permits within the context of this project.

Dynamic Adaptation: Previous works have show the importance of dynamic partitioning and workload rebalancing mechanisms in avoiding inefficiencies due to dynamic behavior changes of the simulation model for both conservative [13, 45] and optimistic [79, 101, 102, 28, 87, 42, 51, 91] synchronization protocols. We differ from these existing studies because they use high-level metrics for detecting workload imbalance,

triggering object migration to other processing elements. In contrast, the tight integration of the cores and low communication latencies in a manycore environment facilitate the tracking of more detailed information and open the door for monitoring object-level behavior, as well as non-local information, to make better informed and coordinated adaptation decisions. This approach allows deeper insight into the role that these dependencies play in affecting performance allowing more precise solutions targeted towards these critical dependencies: this is a level of monitoring and control not investigated by existing work.

Several prior works investigated the dynamic techniques for limiting optimism to an extent that can be justified from the inherent model parallelism [20, 33]. For example, Ferscha proposed an optimism control mechanism which probabilistically delays the execution of scheduled events to avoid potential rollback by maintaining a history record of virtual time differences from the timestamps carried by arriving messages, and forecasting the timestamps of forthcoming messages [33]. As a result, the simulation throttling decision is made based on the use of relatively low-level statistics (such as the estimates of message interarrival times). However, the object-level communication patterns are not considered in this work, nor is the possibility of the use of static analysis to derive these patterns considered.

We believe that this is the first time that the load interference problem is being articulated in the context of PDES simulators. Some related work [79] considers the presence of load and propose solutions to tolerate its effect it using dynamic migration; however, they do not identify the potentially crippling interactions with the operating system scheduler.

6 Proposed Research Schedule

The proposed project duration is four years. Project tasks will be performed on two campuses, SUNY Binghamton (SUNYB) and University of Cincinnati (UC), according to the following schedule:

Year 1: The two teams will carry out a redesign of the simulation kernel for a shared address space multi-threaded environment. The team at SUNYB will perform extensive scalability studies of PDES on manycore platforms with the goal of identifying and quantifying the performance bottlenecks and scalability limits. In addition, the SUNYB team will implement the basic mechanisms to support run-time adaptation, such as dynamic object migration described in Section 3.3. In addition to the simulator development, the team at UC will implement and evaluate relaxed ladder and study alternative scheduling disciplines.

Year 2: SUNYB will investigate the run-time adaptation techniques described in Section 3.3, and their synergy with static analysis. UC will continue its work with manycore PDES, carry out model analysis to isolate exploitable behavior. The two teams will work closely with SUNYB on investigating the synergy of static and run-time analysis. During this year, both teams will also cooperatively work on implementing mechanisms to mitigate the problem of interference from external workload (described in Section 3.5). Starting this year, we extend the simulator architecture models to perform design space exploration of manycores and their impact on PDES and other application performance.

Year 3: We will continue expanding the range of the dependency patterns tracked for the static analysis and dynamic adaptation components of the simulator. The team at UC will investigate techniques for reducing memory pressure. We will design hardware mechanisms to improve the performance of PDES. In this year, we will also develop lightweight consistent checkpointing algorithms to support resilience to failure.

Year 4: In this year, we will finish the implementation and evaluation of the optimized simulator, as well as the static and dynamic adaptation tools. We will evaluate the success of the architecture modifications

in improving the performance of the simulator. All tools and simulators developed during the project will be made available to the public through the project website.

7 Educational and Student Mentoring Activities

Synergistic with the research goals, this project will also include a range of educational activities. Specifically, new graduate-level courses on optimizing fine-grain parallel programs on multicore and manycore systems will be offered in both institutions. In addition, a graduate course on parallel simulation techniques will be offered at SUNY Binghamton and at UC. The PIs have a track record of involving minority students and women in their research project. For example, both Dr. Ponomarev and Dr. Wilsey have advised minority students through the Clark's Fellowship program. All three co-PIs have recruited and advised female students and intend to continue to recruit female students for this project.

In addition, we will continue to actively involve undergraduate students in the project (we have supported 4 different undergraduate students on REUs under our currently supported PDES grant). If the project is funded, we expect to request additional REU supplements to support undergraduate students. A number of small sub-projects, based on the ideas described in this proposal, can be assigned to undergraduate students, providing them the opportunity to work with and optimize real-world large-scale parallel programs. We expect that the undergraduate students will work under the close mentorship of our graduate research assistants.

8 Results from Prior NSF Support

Due to space considerations, we overview the results from the most related grant to the current effort; other recent grants include CNS-1018496, CNS-0958501, CNS-0720811 and CNS-0751161.

Project: CSR:Small:Collaborative Research: Combining Static Analysis and Dynamic Run-time Optimization for Parallel Discrete Event Simulation in Many-Core Environments (CNS-0916323,). This is a collaborative project involving all three PIs. Sept. 1, 2009 – August 31, 2013. Awarded amount: $500,000 (total). The goal of this project is to develop parallel discrete event simulation algorithms that exploit the properties of many-core environments. The supported work has significant intellectual merit and broader impacts as discussed below.

Intellectual Merit: We developed multicore versions of multiple large scale PDES simulators — (WARPED, ROSS and WarpIV) for several emerging multicore and manycore environments (including the AMD Magnycours and the Tilera Tile64 platforms). We identified different bottlenecks for the different platforms and implemented new optimizations to address them. We also developed new partitioning algorithms that use analysis of the model properties to estimate the importance of communication edges and the activity and resource requirements of objects. We evaluated the use of DVFS to improve the performance and energy efficiency of simulation. This project has thus far resulted in seven conference publications [21, 22, 5, 47, 4, 48, 97], which include one paper in IPDPS and four papers in PADS (the top conference in parallel simulation, and as of 2013 the flagship conference of ACM SIGSIM). A number of other papers are under review.

Broader Impacts: The optimized versions of the simulators are available for sharing with the community. The performance analysis and optimizations are applicable to other fine-grained applications. We provided experience and comparative analysis of three modern multicore environments. The partitioning tools are also available for use by the community; they can be modified for other parallel applications. We also received two REU supplement awards (totaling $24,000) and used them to support research with undergraduate students. One of the undergraduate students (Nicole Hofmann) is the second author of [5].

References

[1] N. Abu-Ghazaleh and R. Linderman. Limiting optimism: Time or event count? In *Proc. International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2006.

[2] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. A view of the parallel computing landscape. *Communications of the ACM*, 52(10):56–67, October 2009.

[3] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. Parsec: A parallel simulation environment for complex systems. *Computer*, 31(10):77–85, October 1998.

[4] K. Bahulkar, J. Wang, N. Abu-Ghazaleh, and D. Ponomarev. Partitioning on dynamic bahavior for parallel discrete event simulation. In *26th IEEE/ACM/SCS Workshop on Principles of Advanced and Distributed Simulations (PADS)*, July 2012.

[5] Ketan Bahulkar, Nicole Hofmann, Deepak Jagtap, Nael B. Abu-Ghazaleh, and Dmitry 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.

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

[7] D. Ball and S. Hoyt. The adaptive Time-Warp concurrency control algorithm. In *Distributed Simulation*, pages 174–177. Society for Computer Simulation, January 1990.

[8] 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.

[9] H. Bauer and C. Sporrer. Distributed logic simulation and an approach to asynchronous GVT-calculation. In *6th Workshop on Parallel and Distributed Simulation*, pages 205–208. Society for Computer Simulation, January 1992.

[10] S. Bellenot. Global virtual time algorithms. In *Distributed Simulation*, pages 122–127. Society for Computer Simulation, January 1990.

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

[12] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *Journal of Parallel and Distributed Computing*, pages 207–216, 1995.

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

[14] R. Brown. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, October 1988.

[15] D. Burger and J. Goodman. Memory bandwidth limitations of future microprocessors. In *In Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 78–89, 1996.

[16] 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.

[17] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[18] M. Chetlur, N. Abu-Ghazaleh, R. Radhakrishnan, and P. A. Wilsey. Optimizing communication in Time-Warp simulators. In *12th Workshop on Parallel and Distributed Simulation*, pages 64–71. Society for Computer Simulation, May 1998.

[19] M. Chetlur and P. A. Wilsey. Causality representation and cancellation mechanisms in time warp simulations. In *Proc. International Workshop on Parallel and Distributed Simulation (PADS)*, 2001.

[20] M. Chetlur and P. A. Wilsey. Working set based scheduling in time warp simulation. In *Procedings of the 40th Annual Simulation Symposium*, pages 221–230. Society for Computer Simulation, April 2007.

[21] R. Child and P. A. Wilsey. Dynamically adjusting core frequencies to accelerate time warp simulations in many-core processors. In *Workshop on Principles of Advanced and Distributed Simulation (PADS 12)*, July 2012.

[22] R. Child and P. A. Wilsey. Using DVFS to optimize time warp simulations. In *Proceedings of the 2012 Winter Simulation Conference*, July 2012.

[23] S. Chung and V. Rego. A performance comparison of event calendar algorithms: an empirical approach. *Software Practice and Experience*, 23(10):1107–1138, 1993.

[24] J. Dahl, M. Chetlur, and P. Wilsey. Event list management in distributed simulation. In *Proc. European Parallel Computing Conference (EuroPar)*, 2001.

[25] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. GTW: a Time Warp system for shared memory multiprocessors. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 1332–1339, December 1994.

[26] S. R. Das. Adaptive protocols for parallel discrete event simulation. In *Proc. Winter Simulation Conference*, 1996.

[27] L. M. D'Souza, X. Fan, and P. A. Wilsey. pGVT: An algorithm for accurate GVT estimation. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)*, pages 102–109. Society for Computer Simulation, July 1994.

[28] K. El-Khatib and C. Tropper. On metrics for the dynamic load balancing of optimistic simulations. In *Proc. 32nd Hawaii International Conference on Systems Science (HICCS)*, 1999.

[29] H. Esmaeilzadeh, E. Blem, R. St.-Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 365–376, 2011.

[30] P. Muthana et al. Packaging of multi-core microprocessors: Tradeoffs and potential solutions. In *Proc. Electronic Components and Technology Conference*, 2005.

[31] R Ewald, C Maus, A Rolfs, and A Uhrmacher. Discrete event modelling and simulation in systems biology. *Journal of Simulation*, 1(2):81–96, May 2007.

[32] D. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, pages 306–318, 12 1992.

[33] A. Ferscha. Probabilistic adaptive direct optimism control in time warp. In *Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS 95)*, pages 120–129, June 1995.

[34] A. Ferscha and J. Lüthi. Estimating rollback overhead for optimism control in time warp. In *Annual Simulation Symposium*, April 1995.

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

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

[37] 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)*, pages 149–156, June 1995.

[38] R. M. Fujimoto. Time Warp on a shared memory multiprocessor. *Transactions of Society for Computer Simulation*, pages 211–239, July 1989.

[39] R. M. Fujimoto and M. Hybinette. Computing global virtual time in shared-memory multiprocessors. *ACM Transactions on Modeling and Computer Simulation*, 7(4):425–446, 1997.

[40] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Interscience, January 2000.

[41] Anwar Ghuloum. Face the inevitable, embrace parallelism. *Communications of the ACM*, 52(9):36–38, September 2009.

[42] D. Glazer and C. Tropper. On process migration and load balancing in time warp. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):318–327, 1993.

[43] D. O. Hamnes and A. Tripathi. Investigations in adaptive distributed simulation. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)*, pages 20–23. Society for Computer Simulation, July 1994.

[44] J. Huh, D. Burger, and S.W. 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.

[45] J. Ikonen and J. Porras. Automatic load distribution for conservative distributed simulation. In *Proceedings of the 14th European Simulation Multiconference on Simulation and Modeling*, pages 49–53, 2000.

[46] E. Ipek, S.A. McKee, K. Singh, R. Caruana, B.R. de Supinski, and M. Schulz. Efficient architectural design space exploration via predictive modeling. *ACM Trans. Archit. Code Optim*, 4(4):1–34, 2008.

[47] D. Jagtap, N.Abu-Ghazaleh, and D.Ponomarev. Optimization of parallel discrete event simulator for multi-core systems. In *International Parallel and Distributed Processing Symposium*, May 2012.

[48] Deepak Jagtap, Ketan Bahulkar, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Characterizing and understanding pdes behavior on tilera architecture. In *Workshop on Principles of Advanced and Distributed Simulation (PADS 12)*, July 2012.

[49] 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.

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

[51] M. Jiang, S. Shieh, and C. Liu. Dynamic load balancing in parallel simulation using time warp mechanism. In *Proc. of the International Conference on Parallel and Distributed Systems*, pages 222–229, 1994.

[52] M. G. H. Katevenis. *Reduced Instruction Set Computer Architectures for VLSI*. The MIT Press, 1985.

[53] R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, 2005.

[54] R. Kumar, V. Zyuban, and D.M. 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.

[55] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 3rd edition, 2000.

[56] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the internet's router-level topology. In *Proc. of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2004.

[57] 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)*, pages 211–218, 2007.

[58] M. Liljenstam and R. Ayani. Paritioning PCS for parallel simulation. In *Proc. 5th International Workshop on Modeling Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, January 1997.

[59] Y-B. Lin. Estimating the likelihood of success of lazy cancellation in Time Warp simulations. *International Journal in Computer Simulation*, 1996.

[60] Y-B. Lin and E. D. Lazowska. The optimal checkpoint interval in Time Warp parallel simulation. Technical Report 89–09–04, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, September 1989.

[61] Y-B. Lin and E. D. Lazowska. Processor scheduling for Time Warp parallel simulation. In *Advances in Parallel and Distributed Simulation*, pages 11–14. Society for Computer Simulation, January 1991.

[62] Jason Liu, David M. Nicol, and King Tan. Lock-free scheduling of logical processes in parallel simulation. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation (PADS 2001)*, May 2001.

[63] J. Loew, J. Elwell, D. Ponomarev, and P. Madden. A co-processor approach for accelerating data-structure intensive algorithms. In *IEEE International Conference on Computer Design (ICCD)*, October 2010.

[64] 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.

[65] V. Madisetti, J. Walrand, and D. Messerschmitt. Wolf: A rollback algorithm for optimistic distributed simulation systems. In *Winter Simulation Conference*, pages 296–305. Society for Computer Simulation, December 1988.

[66] F. Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, 18(4):423–434, August 1993.

[67] T. McBrayer and P. A. Wilsey. Process combination to increase event granularity in parallel logic simulation. In *9th International Parallel Processing Symposium*, pages 572–578, April 1995.

[68] R. Merritt. Multicore goals mesh at hot chips. *EETimes Online*, August 2007. Available from http://www.eetimes.com/showArticle.jtml?articleID=201800925.

[69] R. Merritt. X86 cuts to the cores. *EETimes Online*, September 2007. Available from http://www.eetimes.com/showArticle.jtml?articleID=202100022.

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

[71] 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.

[72] Samuel Moore. Multicore is bad news for supercomputers. *IEEE Spectrum Online*, November 2008. Available from http://www.spectrum.ieee.org/nov08/6912.

[73] R. Murphy. On the effects of memory latency and bandwidth on supercomputer application performance. In *Proc. International Symposium on Workload Characterization*, 2007.

[74] R. Noronha and N. B. Abu-Ghazaleh. Active nic optimization for time warp. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[75] A. Palaniswamy and P. A. Wilsey. An analytical comparison of periodic checkpointing and incremental state saving. In *Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS 93)*, pages 127–134. Society for Computer Simulation, July 1993.

[76] A. Palaniswamy and P. A. Wilsey. Scheduling Time Warp processes using adaptive control techniques. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 731–738, December 1994.

[77] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*, 2011.

[78] 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.

[79] 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.

[80] F. Quaglia and V. Cortellessa. Grain sensitive event scheduling in time warp. In *Proc. Workshop on Parallel and Distributed Simulation (PADS)*, 2000.

[81] R. Radhakrishnan, D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. An Object-Oriented Time Warp Simulation Kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, volume LNCS 1505, pages 13–23. Springer-Verlag, December 1998.

[82] R. Radhakrishnan, L. Moore, and P. A. Wilsey. External adjustment of runtime parameters in Time Warp synchronized parallel simulators. In *11th International Parallel Processing Symposium, (IPPS'97)*. IEEE Computer Society Press, April 1997.

[83] R. Radhakrishnan and P. Wilsey. Software control systems for parallel simulation. In *Proc. of the 16th Workshop on Parallel and Distributed Simulation (PADS 02)*, May 2002.

[84] R. Rajan and P. A. Wilsey. Dynamically switching between lazy and aggressive cancellation in a Time Warp parallel simulator. In *Proc. of the 28th Annual Simulation Symposium*, pages 22–30. IEEE Computer Society Press, April 1995.

[85] U. K. V. Rajasekaran, M. Chetlur, G. D. Sharma, R. Radhakrishnan, and P. A. Wilsey. Addressing communication latency issues on clusters for fine grained asynchronous applications — a case study. In *International Workshop on Personal Computer Based Network of Workstations, PC-NOW'99*, April 1999.

[86] P. Reiher and D. Jefferson. Virtual time based dynamic load management in the time warp operating system. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 103–111, 1990.

[87] P. L. Reiher, R. M. Fujimoto, S. Bellenot, and D. Jefferson. Cancellation strategies in optimistic execution systems. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 112–121. Society for Computer Simulation, January 1990.

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

[89] R. Rönngren and R. Ayani. Adaptive checkpointing in Time Warp. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)*, pages 110–117. Society for Computer Simulation, July 1994.

[90] R. Rönngren, J. Riboe, and R. Ayani. Lazy queue: An efficient implementation of the pending-event set. In *Proc. of the 24th Annual Simulation Symposium*, pages 194–204, April 1991.

[91] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer. Dynamic load balancing of a multi-cluster simulator on a network of workstations. In *Proc. Workshop on Parallel and Distributed Simulation (PADS)*, pages 175–180, 1995.

[92] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. MTW: A strategy for scheduling discrete simulation events for concurrent execution. In *Distributed Simulation*, pages 34–42. Society for Computer Simulation, July 1988.

[93] W. T. Tang, R. S M. Goh, and I L-J. Thng. Ladder queue: An o(1) priority queue structure for large-scale discrete event simulation. *ACM Transactions on MOdeling and Computer Simulation*, 15(3):175–204, July 2005.

[94] S.C. Tay, Y.M. Teo, and S.T. Kong. Speculative parallel simulation with an adaptive throttle scheme. In *Proc. of 11th Workshop on Parallel and Distributed Simulation (PADS97)*, pages 116–123, June 1997.

[95] A. I. Tomlinson and V. K. Garg. An algorithm for minimally latent global virtual time. In *Proc of the 7th Workshop on Parallel and Distributed Simulation (PADS)*, pages 35–42. Society for Computer Simulation, July 1993.

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

[97] J. Wang, D. Ponomarev, and N.Abu-Ghazaleh. Performance analysis of multithreaded pdes simulator on multi-core clusters. In *26th IEEE/ACM/SCS Workshop on Principles of Advanced and Distributed Simulations (PADS)*, July 2012.

[98] L. Wang, N. Hoover, E. Porter, and J. Zasio. Ssim: A software levelized compiled-code simulator. In *Proc. Design Automation Conference (DAC)*, pages 2–8, 1987.

[99] Sewook Wee, Jared Casper, Njuguna Njoroge, Yuriy Teslyar, Daxia Ge, Christos Kozyrakis, and Kunle Olukotun. A practical FPGA-based framework for novel CMP research. In *Proceedings of the 15th ACM SIGDA Intl. Symposium on Field Programmable Gate Arrays*, February 2007.

[100] D. Wentzlaff, N. Beckmann, J. Miller, and A. Agarwal. Core count vs cache size for manycore architectures in the cloud. Technical report, MIT CSAIL, February 2010.

[101] L. Wilson and D. Nicol. Automated load balancing in speedes. In *Proc. Winter Simulation Conference*, 1995.

[102] L. Wilson and W. Shen. Experiments in load migration and dynamic load balancing in SPEEDES. In *Proc. of the Winter Simulation Conference*, 1998.

[103] Y. Wiseman and D. Feitelson. Paired gang scheduling. *IEEE Transactions on Parallel and Distributed Systems*, pages 581–592, 2003.

[104] W. Wulf and S. Mckee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23:20–24, 1995.

[105] A. Yoo, M. Jette, and M. Grondona. SLURM: Simple linux utilities for resource management. *Lecture Notes in Computer Science: Job Scheduling Strategies for Parallel Processing*, pages 44–60, 2003.

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