Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

PARALLEL AND DISTRIBUTED SIMULATION

Richard Fujimoto

School of Computational Science & Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA

ABSTRACT

Parallel and distributed simulation is a field concerned with the execution of a simulation program on computing platforms containing multiple processors. This article focuses on the concurrent execution of discrete event simulation programs. The field has evolved and grown from its origins in the 1970's and 1980's and remains an active field of research to this day. An overview of parallel and distributed research is presented ranging from seminal work in the field to address problems such as synchronization to recent work in executing large-scale simulations on supercomputing platforms. Directions for future research in the field are explored.

1 INTRODUCTION

The parallel and distributed simulation field emerged in the 1970's and 1980's from two distinct, overlapping research communities. On the one hand, the *parallel discrete event simulation* (PDES) community was concerned with accelerating the execution of discrete event simulations through the exploitation of high performance computing platforms. In approximately the same time frame the *distributed simulation* community formed growing out of research and development efforts in the defense community that focused on interconnecting separately developed simulations executing on computers interconnected via local and wide area networks. This research initially focused on simulations for training purposes, but quickly broadened to include areas such as analysis and test and evaluation of physical devices. While there are important differences between work in PDES and distributed simulation, there are also many common issues and problems. Here, we informally characterize *parallel and distributed simulation* as a field that encompasses issues arising from both of these communities stemming from the execution of individual simulation programs on platforms ranging from tightly coupled parallel computing platforms or loosely coupled machines connected via a wide area network.

2 PARALLEL DISCRETE EVENT SIMULATION

PDES is concerned with the technologies associated with distributing the execution of a single run of a discrete event simulation program across multiple processors in a high performance computing system. Such platforms include shared-memory multiprocessors and message-based cluster computers. The central goal of PDES is typically to accelerate the execution of the simulation.

A discrete event simulation captures the behavior of an actual or envisioned system over time. Unlike other simulations that operate in a time-stepped fashion and evolve the state of the system from one time step to the next, in a discrete event simulation changes in system state occur at distinct, typically irregular, points in simulation time. A sequential discrete event simulation program includes two fundamental concepts: state variables that capture the state of the system being modeled, and event computations or

simply events that transition the state variables from one state to the next. In a discrete event simulation changes to simulation state occur *only* through event computations. Each event contains a timestamp that represents a point in simulation time at which the state transition occurs. For example, a simulation of an airport might include state variables representing the status of the runway, e.g., idle or busy, as well as other aspects such as the number of aircraft waiting to take off or land. Events might include aircraft arrivals or departures.

A parallel discrete event simulation program can be viewed as a collection of sequential discrete event simulations that interact by exchanging timestamped messages. Each message represents an event that is scheduled (sent) by one simulator to another. Each sequential simulator is referred to as a *logical process* or LP. For example, a simulation of the global air traffic system could be constructed by creating a sequential simulation of each airport and allowing each simulator to schedule events, i.e., send messages, to other simulators. Messages transmitted between airport simulators might represent the arrival of an aircraft flying from one airport to another.

Much of the work in PDES has focused on the synchronization issue. Briefly, the synchronization problem is concerned with managing the execution of a PDES program so that the results that are produced are exactly, or in some cases approximately, equal to a sequential execution of the same program where all events are processed one after the other in timestamp order. A large body of knowledge evolved around different approaches to addressing the synchronization algorithm, and associated algorithms. Other research focused on issues associated with parallel execution such as partitioning simulations, mapping LPs to the processors of a parallel computer, and redistributing computational workloads during program execution to improve efficiency. A number of studies focused on developing techniques to improve the performance of the parallel simulator, e.g., to increase the efficiency of the synchronization algorithm, often in the context of a particular application. Much of this literature is discussed in (Fujimoto 2000) and the references therein.

3 DISTRIBUTED SIMULATION

Distributed simulation is concerned with the execution of simulations over computing platforms that span a much broader geographic extent than parallel computers. In contrast to parallel simulations where the processors executing the simulation reside within a cabinet inside a machine room, a distributed simulation may execute on a set of machines interconnected through a local area network, globally distributed computers communicating via the Internet, or predictive simulations embedded within a physical environment such as a sensor network monitoring traffic in a city. A central object of utilizing distributed simulation systems has been to allow exploitation of geographically distributed resources such as equipment or people in the distributed simulation exercise, or executing simulations in close physical proximity to live data streams to predict future states of an operational systems.

A central issue in and often the principal motivation for utilizing distributed simulation concerns the desire to integrate several different simulators operating on different computing equipment into a single simulation environment. For example, much of the early work in distributed simulations was completed in the defense community for training – tank simulators, flight simulators, computer generated forces, and a variety of other models could be combined to create a distributed virtual environment into which personnel could be embedded to train for hypothetical scenarios and situations.

While synchronization, also known as time management in the distributed simulation literature, is also an issue in distributed simulation, it is not the only issue and typically is not the most important one. Much of the early research in distributed simulation focused on distributing information among the simulators participating in the distributed simulation in an efficient and timely manner. Much research focused on communication protocols such as multicast mechanisms or methods to reduce the amount of data that needed to be communicated through the use of techniques such as dead reckoning (Miller and Thorpe 1995) or data distribution management (Morse and Zyda 2001). Because a large emphasis in distributed simulation has been to achieve interoperability among separately developed simulators, a

substantial amount of effort has focused on developing standards to interconnect simulations. This has resulted in the Distributed Interactive Simulation (DIS) (IEEE Std 1278.1-1995 1995; IEEE Std 1278.2-1995 1995) and the High Level Architecture (HLA) (IEEE Std 1516-2010 2010; IEEE Std 1516.1-2010 2010; IEEE Std 1516.2-2010 2010) standards. Many of the technical issues addressed by the distributed simulation research community are discussed in (Fujimoto 2000; Tolk (ed.) 2012).

The remainder of this article is organized as follows. The next section focuses on conservative algorithms for addressing the synchronization problem. First and second generation algorithms are discussed. Optimistic algorithms are described next, including coverage of the so-called local and global control mechanisms. Finally, future directions for research in the field are discussed, highlighting six areas of research that require further investigation.

4 CONSERVATIVE SYNCHRONIZATION

As discussed earlier, the simulation is assumed to be composed of a set of logical processes (LPs) that communicate by exchanging timestamped event messages. In this section we use the terms events and messages synonymously. A synchronization algorithm is required to ensure that the parallel execution of the simulation produces exactly the same results as a sequential execution on a single processor. In some cases approximate results are acceptable, but the bulk of the research in synchronization algorithms has focused on producing exactly the same results. One can show that this can be achieved by ensuring that each LP processes events in timestamp order.

4.1 First Generation Algorithms

The parallel and distributed simulation field began in the late 1970's with seminal work by Chandy, Misra, and Bryant who defined the synchronization problem and a solution approach (Bryant 1977; Chandy and Misra 1978, 1979). Unlike a timestepped simulation execution where all processors can work on simulation computations in the current time step concurrently, in a PDES program there are typically very few events containing exactly the same time stamp. Thus, one must allow some LPs to advance ahead of others in simulation time in order to capture a sufficient amount of concurrent execution. Space does not permit a full exposition of the rich literature in synchronization algorithms. The following attempts to highlight some of the main technical points. A more lengthy description is presented in (Fujimoto 2000).

The Chandy/Misra/Bryant algorithm uses a mechanism that blocks the execution of an LP until it can guarantee that an event with a smaller timestamp will not later be received. This is accomplished by requiring the messages sent from one LP to another are sent in timestamp order, and the network guarantees ordered delivery, i.e., messages are received in the same order in which they were sent. This ensures events arriving on a single LP-to-LP link are received in timestamp order. Messages arriving on a single link may be stored in a first-in-first-out queue, which will hold messages in timestamp order (see Figure 1(a)). If each FIFO queue contains at least one message, the LP can simply pick the smallest timestamped message, remove it from its queue, and process it. If one or more FIFO queues are empty, the LP must block. Unfortunately, this may lead to deadlock situations where a cycle develops where one LP in the cycle is waiting for the next LP in the cycle (see Figure 1(b)). The CMB algorithm solves this problem by each LP sending *null* messages to neighboring LPs (LPs to which it may send event messages) to provide a guarantee indicating the smallest timestamp value of any message it will later send on that link. To make this guarantee each LP must declare a *lookahead* value. If an LP is currently at simulation time T, and its lookahead value is L, then any message later sent by the LP must have a timestamp of at least $T+L$. One can similarly associate lookahead values with links rather than LPs.

It can be shown that null messages avoid deadlock. However, this algorithm suffers from a problem known as *lookahead creep*. Consider the deadlock shown in Figure 1(b). Note that the smallest timestamped event in this snapshot of the system has a timestamp value of 7. Suppose each LP is at simulation time 5, and has a lookahead of 0.1. Each LP will send a null message to its neighbors

announcing its next message will have a timestamp of at least 5.1 (unless of course a message with a larger timestamp had already been sent), enabling each LP to advance to simulation time 5.1. Each LP will then send another null message with a timestamp of 5.2, enabling the LPs to advance to 5.2. This will repeat until each LP advances to simulation time 7, indicating the message with timestamp 7 can now be processed. Dozens of null messages will have to be sent and processed before the LP can determine that it is safe to process this event. This will clearly be very inefficient.

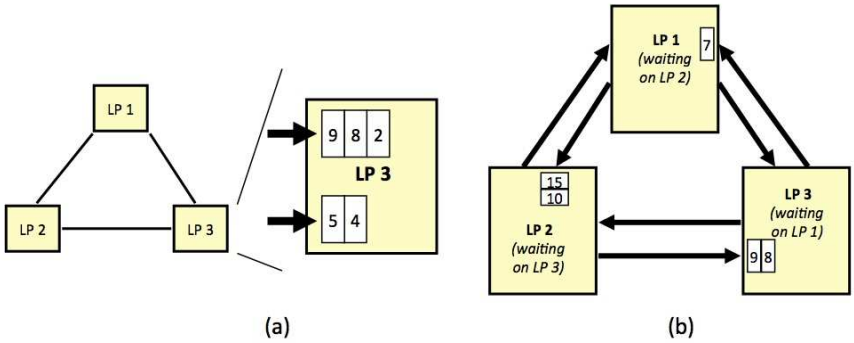

Figure 1: Conservative synchronization. (a) FIFO queues for CMB. (b) A deadlock situation.

4.2 Second Generation Algorithms

Second generation conservative algorithms addressed the lookahead creep problem. The solution to this problem lies in observing that the key piece of information that is required is the timestamp of the next unprocessed event in the system. If the LPs in the previous example knew this information they could immediately advance to simulation time 7 without sending rounds of null messages. Several synchronization algorithms were subsequently proposed that utilize this information, thereby avoiding lookahead creep (Chandy and Misra 1981; Chandy and Sherman 1989; Groselj and Tropper 1988; Lubachevsky 1989; Nicol et al. 1989).

Among these algorithms, we observe that algorithms such as the Bounded Lag (Lubachevsky 1989) and YAWNS (Nicol et al. 1989) algorithms differ significantly from CMB in that unlike CMB they are synchronous algorithms, meaning they utilize global synchronization points (barriers) as a fundamental element of the algorithm. These algorithms divide the computation into a sequence of cycles, or epochs. A global synchronization using a barrier primitive is used to separate the epochs. Each epoch involves (1) determining which events can be safely processed without risk of an LP later receiving a smaller timestamped event, (2) processing these safe events, possibly generating one or more new event messages, and (3) delivering these messages to their destination LPs. The computation repeatedly executes these epochs until the simulation has been completed.

All of the algorithms described above have the property that they strictly avoid allowing an LP to process events out of timestamp order. Algorithms that have this property are referred to as *conservative* synchronization algorithms. Among the many conservative synchronization algorithms that have been developed perhaps the most popular ones today are the original CMB algorithm and YAWNS, no doubt due to their simplicity.

All conservative algorithms rely on a sufficiently large lookahead value to achieve good performance. This is because if there is no lookahead constraint (or equivalently, the lookahead value is zero), the computation for the smallest timestamped event, say 7 in the above example, could in principle send a new event to every other LP in the system with a timestamp of 7. This implies the only concurrent execution that can occur arises from events with exactly the same timestamp. Further, the simulation program must be developed to have good lookahead, i.e., any new event must have a timestamp "reasonably" far into the simulated future. This may be difficult to accomplish for some applications, and can make the code difficult to understand and maintain.

The importance of lookahead is illustrated in Figure 2. This diagram shows a snapshot of a distributed simulation execution where each square box represents and event, plotted according to the timestamp of the event and the LP in which the event occurs. As can be seen, the smallest timestamped event in the system resides in *LP A* and has a timestamp of T_A. In the absence of any lookahead constraints it is possible that processing this event could cause a message with timestamp T_A to be sent to every other LP in the simulation. This implies none of the other events are safe to process because it is possible a smaller timestamped event will later arrive that should be processed first. This suggests that only those events with exactly the same timestamp can be processed concurrently. This severely limits the amount of concurrent processing that can be exploited because most discrete event simulations will have very few events containing exactly the same timestamp. On the other hand, if each LP has a lookahead of L, then the smallest new event that could be generated as a result of processing the event at time T_A must have a timestamp of T_A+L, implying the events shown in red are all safe to process. This example illustrates the importance of lookahead to achieve concurrent processing of events for any conservative synchronization algorithm.

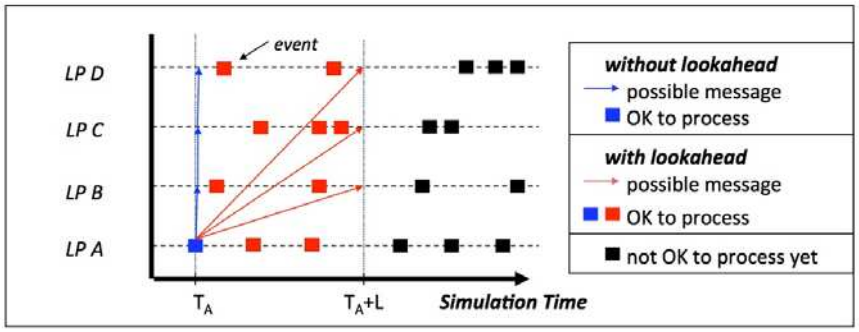

Figure 2: Lookahead. The events in red can be safely processed when exploiting a lookahead.

5 OPTIMISTIC SYNCHRONIZATION

In the early 1980's another approach to addressing the synchronization problem was developed by Jefferson and Sowizral. Their algorithm, called Time Warp, introduced a new approach to addressing the synchronization problem (Jefferson 1985). Unlike conservative approaches that avoid errors, Time Warp allows errors to occur, but the uses a rollback mechanism to recover. Time Warp spawned a number of new algorithms utilizing a rollback mechanism that are now collectively known as optimistic synchronization. The Time Warp algorithm consists of two parts, a local control mechanism and a global control mechanism. Each of these are described next.

5.1 The Local Control Mechanism

Time Warp, does not rely on rules to block the execution of LPs. Rather, LPs are allowed to process events, risking executing events out of timestamp order. When events are processed out of timestamp order, e.g., if an LP processes an event with timestamp 100 and then receives an event from another LP with timestamp 50, a rollback mechanism is invoked to undo the execution of those events processed out of order, i.e., events with timestamp greater than 50.

An event computation can perform two actions that must be undone. It may modify state variables, and it may send event messages to other LPs. Undoing modification of state variables can be accomplished by taking a snapshot of the state of each LP prior to processing each event. This can be done by simply making a copy of the state variables. This approach is called *copy state saving*. This may be expensive in both time to copy the variables, and memory used to hold the saved state. For some applications, state saving overheads may be prohibitive. An alternative approach is to use *incremental state saving* where code is inserted into the event routines to make a copy of a state variable before the

first time it is modified by an event. Unlike copy state saving, incremental state saving requires the address of the variable to stored as well as the copied data. Nevertheless, incremental state saving is much more efficient than copy state saving of an LP requires a large amount of memory to hold state variables, and relatively few variables are modified by each event. A third approach to addressing this problem is to use *reverse computation* where the inverse of the event computation is created and executed to undo an event computation when rollback occurs. For example, the inverse of a computation that increments a state variable is to decrement that variable. This approach has the advantage that it can reduce the amount of computation required in the forward execution of the program which often forms the critical path of the computation and often dictates the execution time, and it can significantly reduce the amount of memory required. However, for many computations the inverse computation cannot be generated. In this case, reverse execution relies on incremental state saving to enable rollback.

The second action that must be undone are message sends performed by the event computation. In contrast to undoing the modification of state variables that are confined to the LP being rolled back, a message sent to another LP may have been processed by that LP, and may have caused the generation of additional messages. These message may, in turn, have been processed resulting in still more messages, and so on. In fact, a single message send may have affected every other LP in the system. A mechanism is required to undo all of these computations! Perhaps the most clever aspect of Time Warp is a very simple mechanism called anti-messages was invented to undo a message send. An anti-message is an identical copy of the original (positive) message that was sent except it has a flag to marking it as an anti-message. To undo a message send, the LP need only send the anti-message to the same destination as the original message. When the anti-message is received by the destination LP, if the original positive message has not yet been processed, then the message/anti-message pair simply annihilate each other and storage for both is reclaimed. If the positive message has already been processed, the LP receiving the anti-message is rolled back to just before the positive message was processed, and then the message/anti-message pair can be annihilated. Rolling back an LP can result in the sending of additional anti-messages, which may in turn cause additional rollbacks. Recursively applying this procedure will erase all of the effects of the original message. The phenomena whereby anti-messages lead to rollbacks in other LPs that may result in the generation of additional anti-messages and still other rollbacks are known as cascaded rollbacks. Clearly cascaded rollbacks are undesirable because they result in the utilization of computation and communication resources without directly performing simulation event computations.

5.2 The Global Control Mechanism

To enable rollback one typically makes a copy of each LP's state variables so an old version of an LP's state can be later restored after a rollback. These check-pointing operations are called *state saves*. This introduces the problem that one must be able to later recover the memory utilized to hold the check-pointed states, as well as any event that has been processed in case it might have to be reprocessed later. A second problem is some computations cannot be rolled back, e.g., I/O operations performed by an LP. Both of these problems are addressed by computing a lower bound on the timestamp of any future rollback that might occur. This computed value is known as *global virtual time* (GVT). Memory used for saved state variables older than GVT (except one, in case a rollback to GVT occurs) can be reclaimed and used for other purposes, and irrevocable operations such as I/O that were performed at a simulation time less than GVT may be performed.

GVT may be defined as the smallest timestamp among those messages (events) and anti-messages that have not yet been processed. Because rollbacks are caused by receiving a message or anti-message in an LP's past, these messages clearly must be considered when determining a lower bound on the timestamp of any future rollback. Therefore, if one could obtain a global snapshot of the computation, GVT could be computed by simply finding the message or anti-message with the smallest timestamp. GVT computation is non-trivial because the GVT computation is typically performed asynchronously "in background" during the execution of the Time Warp simulation.

Several algorithms for computing GVT were developed. Samadi describes two key challenges that must be overcome in computing GVT (Samadi 1985). The first is the *transient message problem*. A transient message is defined as a message that has been sent but has not yet been received. Clearly such messages must be taken into account by the GVT computation. Samadi proposes using message acknowledgements to take these messages into account. The second is referred to as the *simultaneous reporting problem*. This problem arises because different LPs will report their local minimum at different points in wallclock time. Samadi addresses this problem by using a scheme to mark LPs and acknowledgement messages sent by marked LPs. Tha main drawback with Samadi's algorithm is the overhead associated with having the send acknowledgement messages.

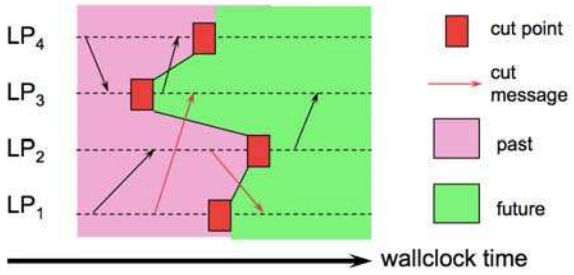

Figure 3: Snapshot of computation used in Mattern's algorithm for computing GVT.

Mattern proposes an elegant algorithm that avoids the need to send message acknowledgements (Mattern 1993). Mattern's algorithm is based developing a distributed snapshot of the execution of the program. Specifically, each LP places a cut point that divides the timeline for the LP's computation into a "past" and "future" part, as shown in Figure 3. Each LP places its cut point asynchronously of the others, so the cut points will in general occur at different points in wallclock time (not to be confused with simulation time). The set of cut points across all LPs forms a *cut* of the computation, and divides the entire distributed computation into a past and future part. A message that is sent in the past part of the computation, and received in the future part is referred to as a *cut message*. There are two cut messages shown in Figure 3. It can be shown that a valid GVT value may be computing by determining the minimum timestamp among (1) any unprocessed message or anti-message within a snapshot of the LP at its cut point and (2) any cut message. Like Samadi's algorithm, Mattern's algorithm must consider transient messages. Rather than using message acknowledgements, each LP keeps a count of the number of messages it has sent and the number of messages it has received. When the sum of these counters over all LPs yield the same value there are no transient messages. Mattern's algorithm utilizes these concepts to efficiently compute GVT. See (Fujimoto 2000) for a more detailed discussion of Mattern's algorithm as well as other algorithms for computing GVT.

5.3 Other Optimistic Algorithms

From a performance standpoint, two central problems faced by optimistic algorithms are an excessive amount of rollback and the time and memory required for state saving. A Time Warp system can suffer from an excessive amount of rolled back computation by, for example, some LPs may advance too far ahead of others leading to long rollbacks as well as excessive memory utilization. Many optimistic algorithms have been proposed to address this issue. Most attempt to limit the amount of optimistic execution, i.e., the amount of event computation that is performed that may be later rolled back. For example, an early technique involves using a sliding window of simulated time and not only allowing events whose timestamp fall beyond this window to be processed (Sokol and Stucky 1990). Another approach delays message sends until it is guaranteed that the send will not be later rolled back, i.e., until GVT advances to the simulation time at which the event was scheduled. This eliminates the need for anti-messages and avoids cascaded rollbacks, i.e., a rollback resulting in the generation of additional rollbacks

(Dickens and Reynolds 1990; Steinman 1992). A number of other techniques have been proposed to improve efficiency by controlling the amount of rollback, or improving the efficiency of the rollback mechanism (Chen and Szymanski 2002, 2003; Fujimoto 1989; Zhang and Tropper 2001). Early approaches to controlling the Time Warp execution used user-defined parameters that had to be tuned to optimize performance. Later work focused on adaptive approaches where the simulation executive automatically monitors the execution and adjusts control parameters to maximize performance. Examples of such adaptive control mechanisms are described in (Das and Fujimoto 1997; Ferscha 1995), among others.

The second problem, memory use in Time Warp, touches upon two main issues. The first concerns the amount of memory required for state saving. The second concerns the amount of time required to perform state saving itself. Increased memory usage can significantly impact performance because hardware caches may not operate very efficiently for programs consuming large amounts of memory. Several techniques have been developed to address this problem. State saving can be performed infrequently rather than before each event computation (Lin et al. 1993; Palaniswamy and Wilsey 1993). An alternative to making a copy of an LP's state variables, referred to as copy state saving, is to use incremental state saving where the original value of a variable is copied just before it is modified by an event computation (Ronngren et al. 1996; West and Panesar 1996). One can roll back computations to reclaim memory resources (Jefferson 1990; Lin and Preiss 1991). The memory used by some state vectors can be reclaimed even though their time stamp is larger than GVT (Preiss and Loucks 1995). Another approach attempts to do away with state saving altogether and instead use reverse execution to undo event computations (Carothers et al. 1999). A reverse execution compiler is used to automatically generate the code that computes the inverse of each event computation, and this inverse code is executed when for each event that must be rolled back. Reverse execution offers the advantage that the burden of executing code to enable rollback, i.e., state saving in conventional Time Warp implementations, is moved to rollback operations rather than the forward execution of events. This makes the overheads associated with state saving less likely to be on the critical path of the computation. In some case, the inverse operation for event computations cannot be created, so this technique typically relies on incremental state saving for such irreversible computations.

Synchronization is a well-studied area of research in the parallel and distributed simulation field. Neither optimistic nor conservative synchronization algorithms dominate the other with respect to performance; indeed, the optimal approach usually depends on the application. In general, if the application has good lookahead and programming the application to exploit this lookahead is not overly burdensome, conservative approaches are the method of choice. Indeed, much research has been devoted to improving the lookahead of simulation applications, e.g., see (Deelman et al. 2001). Otherwise, optimistic synchronization offers greater promise. Disadvantages of optimistic synchronization include the potentially large amount of memory that may be required, and the complexity of optimistic simulation executives. Techniques to reduce memory utilization may further aggravate the complexity issue.

6 DIRECTIONS FOR FUTURE RESEARCH

Below we briefly describe six areas for future research in parallel and distributed simulation.

PDES on Massively Parellal Supercomputers. Over the years numerous successes have been achieved demonstrating the capabilities of PDES technology to accelerate the execution of discrete event simulations. For example, (Fujimoto et al. 2003) examined packet-level simulation of computer communication networks on supercomputers. A metric was defined indicating the number of packet transmissions that could be simulated per second of wallclock time (PTS) by the parallel simulator. Experiments yielding performance as high as 106 million PTS were completed using a conservative synchronization algorithm executing on a supercomputer using 1,536 processors in 2003. By comparison, comparable simulators executing on a sequential machine yielded performance less than 100,000 PTS. In 2007 studies using a synthetic benchmark called PHOLD yielded performance exceeding 529 million

events per second (a packet transmission, discussed earlier, typically consists of two events) using an optimistic synchronization algorithm on a 16,384 processor machine (Perumalla 2007) and in 2009 performance of 12.26 billion events per second using 65,536 processors were achieved (D. W. Bauer Jr. et al. 2009), both using IBM Blue Gene machines. In 2013 Barnes et. al were able to achieve 504 billion events per second using almost 2 million cores of a Blue Gene/Q machine (Barnes et al. 2013). Studies of large-scale simulations of specific applications include epidemic spread (Bisset et al. 2009; Perumalla and Seal 2010) and electromagnetic signal propagation (D. W. Bauer Jr. et al. 2009). These studies yielded event rates *per core* of 138K (2003), 32K (2007), 187K (2009), and 256K (2013) events/second/core, representing only a factor of 2 improvement in single core performance over a span of 10 years. Performance improvements today are being driven almost entirely by increases in parallelism. Processor clock rates have seen only modest increases since 2005 due to physical constraints concerning heat dissipation, resulting in an explosion in the number of cores in supercomputer architectures since 2005. Throughout much of the 1990's and up until 2005 the most powerful supercomputers contained only thousands of cores. The most powerful machines today contain millions.

Exploiting GPU Platforms. A graphics processing unit (GPU) is a hardware accelerator that off-loads computational tasks from the central processing unit (CPU). A principal limitation of GPUs for PDES applications is the single-instruction-stream, multiple-data-stream (SIMD) execution paradigm. GPUs are designed to support data parallel applications where the same operation is performed across multiple data elements. Early work in PDES examined SIMD architectures for queueing network and Ising spin simulations (Lubachevsky 1989) and logic simulation of electronic circuits (Chung and Chung 1991). Work by (Kunz et al. 2012) using GPUs involves sorting events according to event type and clustering their execution to allow SIMD execution. Scheduling and load balancing are discussed in (Romdhanne et al. 2013). Use of GPUs for cellular automata simulations have been explored in application such as traffic simulation (Perumalla et al. 2009; Xu et al. 2014), systems biology (Falk et al. 2011), and Ising Spin (Hawick et al. 2011). In (Park and Fishwick 2011) a time-stepped-like mechanism to implement the discrete event simulation. Another approach utilizing a three step approach – generate events, sort them, process them and use of time parallel simulation techniques is described in (Li, Cai, et al. 2013). In (Liu et al. 2014) a method for hybrid simulation of telecommunication networks is described where a discrete event packet-level simulation is performed by the CPU while a continuous numerical fluid flow simulation of traffic is executed by the GPU.

Exploiting Cloud Computing Platforms. A significant impediment limiting widespread exploitation of PDES technology for large-scale simulation applications has been the need to have access to suitable high performance computing machines. The cloud's "pay-as-you-go" economic model eliminates the need to purchase, operate and maintain high performance computing equipment locally (Fujimoto et al. 2010). Further, by providing parallel and distributed simulation software as a service, cloud computing offers the ability to hide many of the complications of executing parallel and distributed simulation codes from the user, offering the potential to make exploitation of this technology less risky than is the case today. An important issue in cloud computing environments concerns resource sharing. Computing, communications, and I/O resources are shared among many users. Individual users are not guaranteed exclusive access to the processors assigned to that user's virtual cluster. This creates a significant amount of uncertainty and variability in the computation environment on which the parallel or distributed simulation code executes. This variability is cited as a likely cause of performance degradations of conservatively synchronized PDES codes executing on EC2 (Vanmechelen et al. 2012). This can lead to difficulties for parallel simulation applications, especially those that utilize optimistic synchronization techniques. Architectures designed to provide flexible resource sharing are described in (He et al. 2012; Li, Chai, et al. 2013; Liu et al. 2012).

Data Driven Distributed Simulation. Dynamic Data Driven Application Systems (DDDAS) (Darema 2004) are computational systems that dynamically incorporate data from a physical system into executing computations, thereby providing the ability of the application to dynamically steer the measurement

process. These approaches have been widely studied and applied to various science and engineering disciplines for a myriad of purposes. One typical application concerns system monitoring, such as examining the structural and material health (Cortial et al. 2007; Farhat et al. 2006), tracking wildfires (Brun et al. 2012; Douglas et al. 2006; Mandel et al. 2012; Mandel et al. 2005) and hurricanes (Allen 2007), or prediction and tracking regional scale weather phenomena (Plale et al. 2005). A second use concerns optimizing the operations of a physical system. For example, in an emergency situation alternate evacuation scenarios may be modeled and evaluated in order to minimize evacuation time (Chaturvedi et al. 2006). The evacuation plan may need to adapt as the evacuation evolves when unforeseen events arise (Chaturvedi et al. 2006). Additional examples include path planning for unmanned aerial vehicles (Kamrani and Ayani 2007; Madey et al. 2012), tuning parameters for computer networks (Ye et al. 2008), managing semiconductor manufacturing systems (Low et al. 2005), and managing surface transportation systems to mitigate congestion (Fujimoto et al. 2007; Suh et al. 2014). They have been proposed for decision support systems for manufacturing applications, as described in (Lendermann et al. 2005). An approach to distributed, online simulations of queueing networks is discussed in (Huang et al. 2010).

Power and Energy Consumption. Power consumption has become a major concern for many parallel and distributed computing applications. The need to reduce power consumption is clear in mobile and embedded computing where reduced power consumption can result in increase battery life or enable the use of smaller batteries thereby reducing the size and weight of devices. In high-end computing power consumption is a dominant cost associated with operating large data centers and supercomputers, and a substantial amount of effort has gone into developing techniques to reduce these costs. So far, work in power and energy aware computing has largely focused on low-level aspects of the computing system in terms of effectively utilizing specific hardware capabilities and the development of operating systems and compilers to reduce energy usage consistent with performance constraints. The design of a simulation, e.g., in terms of data structures and algorithms with respect to energy and power consumption has not been extensively studied. The relationships among model detail and fidelity and power consumption are not well understood. These tradeoffs must consider analysis of data produced by the simulation, which will likely be greater for more detailed model, in addition to the execution of the simulation model. Further, modeling results must be produced in a timely fashion when the model is used for managing operational systems, imposing real-time constraints on model execution time. The frequency and amount of data utilized by the simulation will have a large impact on power consumption and energy use. In addition to the power required to transmit the data, data analysis algorithms will also consume energy to process incoming data. Further, synchronization algorithms can significantly affect the amount of energy that is consumed.

Interoperable Models. In (Petty and Weisel; 2003) composability is defined as "the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements." Composability is not a binary, yes/no property, but rather is characterized by degrees. Three levels of integration are proposed in (Page et al. 2004), and refined in (Tolk 2012) to define seven levels that highlight several key issues to achieving composeability. A longstanding goal of the distributed simulation community has been to achieve "plug-and-play" interoperability where one is able to automatically compose separately developed simulations independent of the context into which they are embedded. This implies model composeability, and is far from the capabilities of current technologies. Multi-modeling or multi-paradigm modeling addresses the problem of constructing heterogeneous models of systems where the components of the system are modeled using different modeling approaches. Multi-modeling has been widely used in embedded system design and cyber-physical systems where models, computations, and networks are combined with physical devices. Computer Automated Multi-Paradigm Modeling (CAMPaM) strives to create domain-independent frameworks to support the development of heterogeneous models (Mosterman and Vangheluwe 2004). These systems must address issues concerned with integration of multiple modeling formalisms, integrations of models at different levels of abstraction, and metamodels. A metamodel

defines the modeling language. The same metamodeling language can describe multiple modeling languages, and thus can be used to specify systems that utilize multiple formalisms (Cetinkaya and Verbraeck 2011; Vangheluwe and Lara 2002). The metamodel provides a means of describing the overall system composed of the different constituent modeling formalisms supported by the modeling system. Compilers or translators provide a means to automatically transform a metamodel for a system into a simulation model utilizing multiple formalisms. Finally, the metmodeling language itself can be described by a meta-metamodel language. Ptolemy II and AToM3 are examples of multi-modeling frameworks (De Lara and Vangheluwe 2002; Ptolemaeus (ed.) 2014).

7 CONCLUSIONS

Parallel and distributed simulation continues to be an important field of research. A rich body of literature has developed addressing problems such as synchronization to ensure correct results are produced. The field continues to evolve, driven by requirements derived from application and changes in the underlying computing platforms. Future research will need to address requirements and constraints derived from applications as well as major changes in the underlying computing platform and software.

ACKNOWLEDGMENTS

Support for Fujimoto's research in parallel and distributed simulation has been provided by the National Science Foundation under Grant 1441208 and the Air Force Office of Scientific Research under Grant FA9550-13-1-0100.

REFERENCES

Allen, G. 2007. "Building a Dynamic Data Driven Application System for Hurricane Forecasting." In *Computational Science – ICCS 2007*, edited by Y. Shi, G. D. v. Albada, J. Dongarra, and P. M. A. Sloot, 1034–1041. Berlin: Springer Berlin Heidelberg.

Barnes, P. D., C. D. Carothers, D. R. Jefferson, and J. M. LaPre. 2013. "Warp Speed: Executing Time Warp on 1,966,080 Cores." In *Principles of Advanced Discrete Simulation*, 327-336.

Bauer Jr., D. W. C. D. Carothers, and A. Holder. 2009. "Scalable Time Warp on Blue Gene Supercomputers." In *Principles of Advanced and Distributed Simulation*, 35-44.

Bisset, K. R., J. Chen, X. Feng, V. S. A. Kumar, and M. V. Marathe. 2009. "Epifast: A Fast Algorithm for Large Scale Realistic Epidemic Sim- Ulations on Distributed Memory Systems." In *International Conference of Supercomputing*, 430–439.

Brun, C., T. Artés, T. Margalef, and A. Cortés. 2012. "Coupling Wind Dynamics into a DDDAS Forest Fire Propagation Prediction System." In *Proceedings of the International Conference on Compuational Science.*

Bryant, R. E. 1977. "Simulation of Packet Communications Architecture Computer Systems." In *MIT-LCS-TR-188*.

Carothers, C. D., K. Perumalla, and R. M. Fujimoto. 1999. "Efficient Optimistic Parallel Simulation Using Reverse Computation." *ACM Transactions on Modeling and Computer Simulation* 9(3):224-253.

Cetinkaya, D., and A. Verbraeck. 2011. "Metamodeling and Model Transformations in Modeling and Simulation." In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 3048-3058. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Chandy, K. M., and J. Misra. 1979. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." *IEEE Transactions on Software Engineering* SE-5 (5):440-452.

Chandy, K. M., and J. Misra. 1981. "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations." *Communications of the ACM* 24 (4):198-205.

Chandy, K. M., and R. Sherman. 1989. "The Conditional Event Approach to Distributed Simulation." In *Proceedings of the SCS Multiconference on Distributed Simulation*, 93-99.

Chaturvedi, A., A. Mellema, S. Filatyev, and J. Gore. 2006. "DDDAS for Fire and Agent Evacuation Modeling of the Rhode Island Nightclub Fire." In *Computational Science – ICCS 2006*, edited by V. N. Alexandrov, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, 433–439. Berlin: Springer.

Chen, G., and B. K. Szymanski. 2002. "Lookback: A New Way of Exploiting Parallelism in Discrete Event Simulation." In *the 16th Workshop on Parallel and Distributed Simulation*, 153-162.

Chen, G., and B. K. Szymanski. 2003. "Four Types of Lookback." In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, 3-10.

Chung, M., and Y. Chung. 1991. "An Experimental Analysis of Simulation Clock Advancement in Parallel Logic Simulation on an Simd Machine." In *Advances in Parallel and Distributed Simulation*, 125-132. SCS Simulation Series.

Cortial, J., C. Farhat, L. J. Guibas, and M. Rajashekhar. 2007. "Compressed Sensing and Time-Parallel Reduced-Order Modeling for Structural Health Monitoring Using a Dddas." In *Computational Science – ICCS 2007*, edited by Y. Shi, G. D. v. Albada, J. Dongarra, and P. M. A. Sloot, 1171–1179. Berlin: Springer.

Darema, F. 2004. "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements." In *International Conference on Computational Science*.

Das, S. R., and R. M. Fujimoto. 1997. "Adaptive Memory Management and Optimism Control in Time Warp." *ACM Transactions on Modeling and Computer Simulation* 7 (2):239-271.

De Lara, J., and H. Vangheluwe. 2002. "Atom3: A Tool for Multi-Formalism and Meta-Modelling." In *Proceedings of the Fundamental Approaches to Software Engineering*, edited by R.-D. Kutsche, and H. Weber, 174–188. Springer.

Deelman, E., R. Bagrodia, R. Sakellariou, and V. Adve. 2001. "Improving Lookahead in Parallel Discrete Event Simulations of Large-Scale Applications Using Compiler Analysis." In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, 5-13.

Dickens, P. M., and J. Reynolds, P. F. 1990. "Srads with Local Rollback." In *Proceedings of the SCS Multiconference on Distributed Simulation*, 161-164.

Douglas, C. C., R. A. Lodder, J. D. Beezley, J. Mandel, R. E. Ewing, Y. Efendiev, G. Qin, M. Iskandarani, J. Coen, A. Vodacek, M. Kritz, and G. Haase. 2006. "DDDAS Approaches to Wildland Fire Modeling and Contaminant Tracking." In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 2117-2124. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Falk, M., M. Ott, T. Ertl, M. Klann, and H. Koeppl. 2011. "Parallelized Agent-Based Simulation on Cpu and Graphics Hardware for Spatial and Stochastic Models in Biology." In *9th International Conference on Computational Methods in Systems Biology*, 73-82.

Farhat, C., J. G. Michopoulos, F. K. Chang, L. J. Guibas, and A. J. Lew. 2006. "Towards a Dynamic Data Driven System for Structural and Material Health Monitoring." In *Computational Science – ICCS 2006*, edited by V. N. Alexandrov, G. D. v. Albada, P. M. A. Sloot, and J. Dongarra, 456–464. Berlin: Springer Berlin Heidelberg.

Ferscha, A. 1995. "Probabilistic Adaptive Direct Optimism Control Iin Time Warp." In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 120-129.

Fujimoto, R. M. 1989. "Time Warp on a Shared Memory Multiprocessor." *Transactions of the Society for Computer Simulation* 6 (3):211-239.

Fujimoto, R. M. 2000. *Parallel and Distributed Simulation Systems*: Wiley Interscience.

Fujimoto, R. M., M. Hunter, J. Sirichoke, M. Palekar, H.-K. Kim, and W. Suh. 2007. Ad Hoc Distributed Simulations. In *Principles of Advanced and Distributed Simulation*.

Fujimoto, R. M., A. W. Malik, and A. J. Park. 2010. "Parallel and Distributed Simulation in the Cloud." *SCS Modeling and Simulation Magazine, Society for Modeling and Simulation, Intl.* 1 (3).

Fujimoto, R. M., K. Perumalla, A. Park, H. Wu, M. H. Ammar, and G. F. Riley. 2003. Large-Scale Network Simulation: How Big? How Fast? In *Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*.

Groselj, B., and C. Tropper. 1988. "The Time of Next Event Algorithm." In *Proceedings of the Scs Multiconference on Distributed Simulation*, 25-29. Society for Computer Simulation.

Hawick, K. A., A. Leist, and D. P. Playne. 2011. "Regular Lattice and Small- World Spin Model Simulations Using Cuda and Gpus." *International Journal of Parallel Programming* 39 (2):183–201.

He, H., R. Li, X. Dong, Z. Zhang, and H. Han. 2012. "An Efficient and Secure Cloud-Based Distributed Simulation System." *Journal of Applied Mathematics & Information Sciences* 6 (3):729-736.

Huang, Y.-L., M. Hunter, C. Alexopoulos, and R. M. Fujimoto. 2010. "Ad Hoc Distributed Simulation of Queueing Networks." In *Principles of Advanced and Distributed Simulations*.

IEEE Std 1278.1-1995. 1995. *IEEE Standard for Distributed Interactive Simulation -- Application Protocols*. New York, NY: Institute of Electrical and Electronics Engineers, Inc.

IEEE Std 1278.2-1995. 1995. *IEEE Standard for Distributed Interactive Simulation -- Communication Services and Profiles*. New York, NY: Institute of Electrical and Electronics Engineers Inc.

IEEE Std 1516-2010. 2010. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*. New York, NY: Institute of Electrical and Electronics Engineers, Inc.

IEEE Std 1516.1-2010. 2010. *IEEE Standard for Modeling and Simulation High Level Architecture (HLA) - Interface Specification*. New York, NY: Institute of Electrical and Electronics Engineers, Inc.

IEEE Std 1516.2-2010. 2010. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Object Model Template (OMT) Specification*. New York, NY: Institute of Electrical and Electronics Engineers, Inc.

Jefferson, D. 1985. "Virtual Time." *ACM Transactions on Programming Languages and Systems* 7 (3):404-425.

Jefferson, D. R. 1990. "Virtual Time Ii: Storage Management in Distributed Simulation." In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, 75-89.

Kamrani, F., and R. Ayani. 2007. Using on-Line Simulation for Adaptive Path Planning of UAVs. *IEEE International Symposium on Distributed Simulation and Real-Time Applications*.

Kunz, G., D. Schemmel, J. Gross, and K. Wehrle. 2012. "Multi-Level Parallelism for Time and Cost Efficient Parallel Discrete Event Simulation on Gpus." In *Principles of Advanced and Distributed Simulation*, 23-32.

Lendermann, P., M. Y. H. Low, B. P. Gan, N. Julka, L.-P. Chan, L. H. Lee, S. J. E. Taylor, S. J. Turner, W. Cai, X. Wang, T. Hung, L. F. McGinnis, and S. Buckley. 2005. "An Integrated and Adaptive Decision-Support Framework for High-Tech Manufacturing and Service Networks." In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 2052-2062. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Li, B.-H., X. Chai, B. Hou, C. Yang, T. Li, T. Lin, Z. Zhang, Y. Zhang, W. Zhu, and Z. Zhao. 2013. "Research and Application on Cloud Simulation." In *Summer Computer Simulation Conference*, 157-170.

Li, X., W. Cai, and S. Turner. 2013. "Gpu Acceleratedthree-Stageexecution Model for Event-Parallel Simulation." In *Principles of Advanced Discrete Simulation*, 57-66.

Lin, Y.-B., and B. R. Preiss. 1991. "Optimal Memory Management for Time Warp Parallel Simulation." *ACM Transactions on Modeling and Computer Simulation* 1 (4).

Lin, Y.-B., B. R. Preiss, W. M. Loucks, and E. D. Lazowska. 1993. "Selecting the Checkpoint Interval in Time Warp Simulations." In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, 3-10.

Liu, J., Y. Liu, Z. Du, and T. Li. 2014. "GPU-Assisted Hybrid Network Traffic Model." In *Principles of Advanced Discrete Simulation*, 63-74.

Liu, X., X. Qiu, B. Chen, and K. Huang. 2012. "Cloud-Based Simulation: The State-of-the-Art Computer Simulation Paradigm." In *Principles of Advanced and Distributed Simulation*, 71-74.

Low, M. Y. H., K. W. Lye, P. Lendermann, S. J. Turner, R. T. W. Chim, and S. H. Leo. 2005. "An Agent-Based Approach for Managing Symbiotic Simulation of Semiconductor Assembly and Test Operation." In *Proceedings of the 14th International Joint Conference on Autonomous Agents and Multiagent Systems*, 85–92. New York: Association for Computing Machinery, Inc.

Lubachevsky, B. D. 1989. "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks." *Communications of the ACM* 32 (1):111-123.

Madey, G. R., M. B. Blake, C. Poellabauer, H. Lu, R. R. McCune, and Y. Wei. 2012. Applying Dddas Principles to Command, Control and Mission Planning for UAV Swarms. *Proceedings of the International Conference on Compuational Science*.

Mandel, J., J. D. Beezley, A. K. Kochanski, V. Y. Kondratenko, and M. Kim. 2012. Assimilation of Perimeter Data and Coupling with Fuel Moisture in a Wildland Fire – Atmosphere DDDAS. In *International Conference on Compuational Science*.

Mandel, J., L. S. Bennethum, M. Chen, J. L. Coen, C. C. Douglas, L. P. Franca, C. J. Johns, M. Kim, A. V. Knyazev, R. Kremens, V. Kulkarni, G. Qin, A. Vodacek, J. Wu, W. Zhao, and A. Zornes. 2005. "Towards a Dynamic Data Driven Application System for Wildfire Simulation." In *Computational Science – ICCS 2005*, edited by V. S. Sunderam, G. D. v. Albada, P. M. A. Sloot, and J. J. Dongarra, 632–639. Berlin: Springer Berlin Heidelberg.

Mattern, F. 1993. "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation." In *Journal of Parallel and Distributed Computing*.

Miller, D. C., and J. A. Thorpe. 1995. "Simnet: The Advent of Simulator Networking." *Proceedings of the IEEE* 83 (8):1114-1123.

Morse, K. L., and M. Zyda. 2001. "Multicast Grouping for Data Distribution Management." *SIMPRA - Journal of Simulation Practice and Theory* Fall (Elsevier).

Mosterman, P., and H. Vangheluwe. 2004. "Computer Automated Multi-Paradigm Modeling: An Introduction." *Simulation: Transactions of The Society for Modeling and Simulation International* 80 (9):433-450.

Nicol, D. M., C. Micheal, and P. Inouye. 1989. "Efficient Aggregaton of Multiple Lps in Distributed Memory Parallel Simulations." In *Proceedings of the 1989 Winter Simulation Conference*, edited by E. A. MacNair, K. J. Musselman, and P. Heidelberger, 680-685. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Page, E. H., R. Briggs, and J. A. Tufarolo. 2004. "Toward a Family of Maturity Models for the Simulation Interconnection Problem." In *Spring 2004 Simulation Interoperability Workshop*.

Palaniswamy, A. C., and P. A. Wilsey. 1993. "An Analytical Comparison of Periodic Checkpointing and Incremental State Saving." In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, 127-134.

Park, H., and P. Fishwick. 2011. "An Analysis of Queuing Network Simulation Using Gpu-Based Hardware Acceleration." *ACM Transactions on Modeling and Computer Simulation* 21 (3).

Perumalla, K. S. 2007. "Scaling Time Warp-Based Discrete Event Execution to 10**4 Processors on a Blue Gene Supercomputer." In *Proceedings of the ACM Computing Frontiers Conference*.

Perumalla, K. S., G. A. Brandon, B. Y. Srikanth, and K. S. Sudip. 2009. "Gpu-Based Real-Time Execution of Vehicular Mobility Models in Large-Scale Road Network Scenarios." In *Principles of Advanced and Distributed Simulation*, 95-103.

Perumalla, K. S., and S. K. Seal. 2010. "Reversible Parallel Discrete-Event Execution of Large-Scale Epidemic Outbreak Models." In *Principles of Advanced and Distributed Simulation*, 106-113.

Petty, M. D., and E. W. Weisel;. 2003. "A Composability Lexicon." In *IEEE Spring Simulation Interoperability Workshop*.

Plale, B., D. Gannon, and D. Reed. 2005. Towards Dynamically Adaptive Weather Analysis and Forecasting in Lead. *International Conference on Computational Science*.

Preiss, B. R., and W. M. Loucks. 1995. "Memory Management Techniques for Time Warp on a Distributed Memory Machine." In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 30-39.

Ptolemaeus (ed.), C. 2014. *System Design, Modeling, and Simulation Using Ptolemy II*: Ptolemy.org

Romdhanne, B. B., M. S. M. Bouksiaa, N. Nikaein, and C. Bonnet. 2013. "Hybrid Scheduling for Event-Driven Simulation over Heterogeneous Computers." In *Principles of Advanced Discrete Simulation*, 47-56.

Ronngren, R., M. Liljenstam, J. Montagnat, and R. Ayani. 1996. "Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation." In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, 70-77.

Samadi, B. 1985. "Distributed Simulation, Algorithms and Performance Analysis." Computer Science Department, University of California, Los Angeles, Los Angeles, California.

Sokol, L. M., and B. K. Stucky. 1990. "MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm." In *Proceedings of the SCS Multiconference on Distributed Simulation*, 169-173.

Steinman, J. S. 1992. "Speedes: A Multiple-Synchronization Environment for Parallel Discrete Event Simulation." *International Journal on Computer Simulation*:251-286.

Suh, W., M. Hunter, and R. M. Fujimoto. 2014. "Ad Hoc Distributed Simulation for Transportation System Monitoring and near-Term Prediction." *Simulation Modeling Practice and Theory* 41:1-14.

Tolk (ed.), A. 2012. *Engineering Principles of Combat Modeling and Distributed Simulation*. Hoboken, New Jersey: John Wiley and Sons

Tolk, A. 2012. "Challenges of Distributed Simulation." In *Engineering Principles of Combat Modeling and Distributed Simulation*, edited by A. Tolk. John Wiley and Sons Inc.

Vangheluwe, H., and J. D. Lara. 2002. "Meta-Models Are Models Too." In *Proceedings of the 2002 Winter Simulation Conference*, edited by E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, 597 – 605. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Vanmechelen, K., S. De Munck, and J. Broeckhove. 2012. "Conservative Distributed Discrete Event Simulation on Amazon EC2." In *International Symposium on Cluster, Cloud, and Grid Computing*, 853-860.

West, D., and K. Panesar. 1996. "Automatic Incremental State Saving." In *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, 78-85.

Xu, Y., G. Tan, X. Li, and X. Song. 2014. "Mesoscopic Traffic Simulation on Cpu/Gpu." In *Principles of Advanced Discrete Simulation*, 39-49.

Ye, T., H. Kaur, S. Kalyanaraman, and M. Yuksel. 2008. "Large-Scale Network Parameter Configuration Using an on-Line Simulation Framework." *IEEE/ACM Transactions on Networking* 16:777–790.

Zhang, J. L., and C. Tropper. 2001. "The Dependence List in Time Warp." In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, 35-45.

AUTHOR BIOGRAPHIES

RICHARD FUJIMOTO is Regents' Professor in the School of Computational Science and Engineering at the Georgia Institute of Technology. He received a Ph.D. in Computer Science & Electrical Engineering from the University of California-Berkeley in 1983. His email address is fujimoto@cc.gatech.edu.