Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

POWER CONSUMPTION IN PARALLEL AND DISTRIBUTED SIMULATIONS

Richard M. Fujimoto

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

ABSTRACT

The energy and power consumed by computing applications have long been important concerns in mobile systems and have recently become of great interest in high performance and cloud computing. To date, only a limited amount of work has considered power consumption in parallel and distributed simulation systems. A variety of options to reduce power consumption in these systems are discussed, suggestive of directions for future research in this increasingly important area.

1 INTRODUCTION

Energy consumption has long been a major concern in mobile computing systems. Reduced energy consumption can lead to longer times between recharging and/or the ability to use smaller, lighter weight batteries. Executing parallel and distributed simulations on mobile computing platforms is an area of increasing interest in the context of symbiotic simulations (Fujimoto et al. 2002) or dynamic data-driven application systems (DDDAS) (Darema 2004). Such systems involve incorporating live data from instrumented systems into simulations in order to optimize the system and/or steer the measurement process. Trends such as edge computing are pushing computations away from centralized computing facilities into devices embedded in and interacting with the real world. Placing the simulations in close proximity to the physical system, e.g., within a sensor network, reduces or eliminates reliance on connectivity to the central command center, offers the potential for greater scalability, enables faster response time for latency-critical applications, and avoids privacy issues associated with storing data in a centralized facility. In situations such as these the simulation may need to operate on battery-operated mobile devices, so energy consumption becomes an important concern.

For example, distributed simulations can be used to create adaptive sensor networks to monitor dynamically changing physical systems. A collection of small, battery-operated unmanned aerial vehicles (UAVs) might be assigned the task of monitoring a physical system, e.g., tracking the flow of traffic in a city, monitoring the spread of a forest fire, or assessing the dispersion of a hazardous chemical plume following an accident (Fujimoto et al. 2007; Kamrani and Ayani 2007; Madey et al. 2012). Each UAV is equipped with sensors, an on-board computer, and wireless communications, and assigned to collect information in a certain geographical area. Collectively the team of UAVs may then execute a distributed simulation to project the future state of the system, e.g., to predict the evolution of the fire in order to determine how best to relocate the UAVs in order to continue monitoring its spread.

In other contexts it may be more appropriate to utilize remote servers, e.g., computation resources available in the cloud, to execute the simulations. Here the sensor nodes are only used for data collection and aggregation. Indeed, this approach is more commonly used today. Use of remote servers may be preferable if the embedded computational resources are insufficient to complete the simulations in a timely fashion, the latency associated with communications to and from the remote servers is acceptable, and use of aggregated data feeds yields sufficiently accurate simulation results. Here, the power utilized for communication with battery-powered nodes as well as that consumed by the remote server systems, as discussed momentarily, may be of greater concern.

In high performance computing power consumption is a major impediment to achieving increased levels of performance due to limitations in dissipating heat from electronic circuits. It has been cited as a key hurdle in achieving exascale performance in supercomputers. The Thermal Design Power (TDP) is the amount of heat generated by a computer chip or component during normal operation for which the cooling system has been designed (Huck 2011). Power capping used in some systems places a maximum amount of power that can be consumed by a computer server. A typical goal for an HPC application might be to minimize execution time subject to staying within the specified power cap constraint.

Further, power consumption in supercomputers and data centers used for cloud computing applications is a major operating expense. The U.S. Department of Energy has set a goal of 20 megawatts as the maximum power consumed by an exascale supercomputer. Electricity is a major cost in operating data centers. It is estimated that in total, data centers consumed approximately 70 billion kW-hours or about 1.8% of the total electricity consumption in the U.S. in 2014 (Shehabi et al. 2016). As such, reducing power consumption is increasing in importance for high performance and cloud computing applications.

2 PARALLEL AND DISTRIBUTED SIMULATION

We first review key concepts and algorithms in parallel and distributed simulation that will be utilized later. Further discussion of this topic can be found in (Fujimoto 2000). *Parallel simulation* is concerned with distributing the execution of a simulation program across the processors in a tightly coupled multiprocessor. The principle goal is usually to reduce execution time. Here, we are primarily concerned with discrete event simulations. Parallel execution is accomplished by partitioning the system being modeled into a set of physical processes, and modeling each physical process with a simulation program referred to as a logical process (LP). LPs communicate exclusively by exchanging timestamped events or messages. Each event/message contains a timestamp with a value in simulation time indicating when that event would occur in the physical system. Sending a message from one LP to another is equivalent to the sending LP scheduling a new event at the receiving LP. The state variables making up the simulation are partitioned among the LPs; no shared state among the LPs is allowed. The parallel simulator is often partitioned into two components – the simulation engine that manages the execution of the simulation and is independent of the particular simulation application being studied, and the simulation application that includes the model where all aspects related to the domain of application are represented.

Distributed simulation typically refers to the execution of the simulation on a loosely coupled distributed computing platform such as processors interconnected by a local or wide area network. While scalability and performance is often a goal or requirement in distributed simulations, often a more compelling objective is to interconnect or federate separately developed simulators, thereby realizing great cost savings via software reuse compared to developing new simulation models. A key challenge is to allow separately developed simulators to interoperate and exchange data. Standards such as the High Level Architecture (IEEE Std 1516.3-2010 2010) and Distributed Interactive Simulation (IEEE Std 1278.1-1995 1995) have been developed to facilitate interoperability. Middleware often referred to as Runtime Infrastructure (RTI) software implements services for simulations to exchange data, synchronize, and manage the execution of the distributed simulation. Like the simulation engine in a parallel simulator, RTI software is often independent of the simulation application being studied.

Parallel and distributed simulations are different, but share many aspects in common. In the following, when the distinction is not important, we simply use the term “distributed simulation” to generically refer to both.

2.1 Synchronization

Parallel and distributed simulations that are used for system analysis (as opposed to training) utilize mechanisms to ensure that the parallel/distributed execution yields the same results as a sequential execution. In discrete event simulations this is accomplished by ensuring that the events processed within

each LP are processed in timestamp order. Provided events with the same timestamp are processed in the same order by the parallel/distributed and sequential simulation, ensuring each LP processes events in timestamp order is sufficient to ensure that the same results as a sequential simulation are produced.

There are two major categories of synchronization algorithms used to ensure each LP processes events in timestamp order. The first, called *conservative synchronization*, ensures that each LP never processes an event until it can guarantee that it will not later receive an event containing a smaller timestamp. This can be accomplished by each LP determining a *lower bound on timestamp* (LBTS) of any message it might later receive. The earliest algorithms developed independently by Chandy and Misra (Chandy and Misra 1979) and Bryant (Bryant 1977) are known as the null message or Chandy/Misra/Bryant (CMB) algorithm. CMB has LPs send dummy or "null" messages to each other indicating a lower bound on the time stamp of any message it might send in the future. This lower bound is computed using the LP's current simulation time, and the minimum amount of simulation time into the future that could be used to generate a new message. This latter value is referred to as the LP's *lookahead*. Lookahead is a fundamental requirement of all conservative simulations, and a large lookahead value relative to the average amount of simulation time between events in an LP is usually required to achieve efficient execution. Second generation conservative algorithms also utilized the timestamp of the next unprocessed event within each LP to compute LBTS values. This can greatly improve the efficiency of the algorithm, but large lookahead values are still required.

The other major approach is called *optimistic synchronization*. Jefferson's Time Warp algorithm was the first, and remains the most well-known optimistic algorithm (Jefferson 1985). While conservative algorithms avoid synchronization errors, i.e., processing events within an LP out of timestamp order, optimistic algorithms use a detection and recover approach. An LP can easily detect when it has processed events out of timestamp order by simply comparing the timestamp of each incoming message with that of the last event it processed. The computations associated with events with timestamp larger than the incoming message must be rolled back. Rolling back an event involves restoring the state variables of an LP to that which existed prior to processing the event, and "unsending" any messages sent by the rolled back event. The latter is accomplished in Time Warp using a mechanism called anti-messages that cancel previously sent messages. Anti-messages may cause additional rollbacks, leading to the possibility of rollback cascades where one rollback results in a second, or third, etc. Time Warp also requires the computation of a value called *Global Virtual Time* (*GVT*) which is a lower bound on the timestamp of future rollbacks. GVT is required to reclaim memory, e.g., snapshots of LP state variables needed because of the possibility of rollback, and to perform operations such as I/O that cannot be rolled back. A number of other optimistic synchronization algorithms were developed after Time Warp appeared, however, most rely on the basic mechanisms used by Time Warp described above.

From the standpoint of power and energy consumption, synchronization is important because it represents overheads required by the distributed simulation, separate from the simulation application. The amount of power and energy required for synchronization can be a significant concern in parallel and distributed simulations. More will be said about this later.

2.2 Data Distribution

A second major function in distributed simulation concerns the distribution of information among the LPs or federates making up the simulation. Data distribution in parallel discrete event simulations is usually relatively straightforward because the parallel simulator is developed as a single unified body of code so the sending LP knows which other LP(s) should receive the messages it is sending. In federated distributed simulations this is not so straightforward because each federate is designed to be an autonomous simulator that can interoperate with other simulations that were developed separately from each other. When the state of a federate such as the position of a moving vehicle modeled by that federate changes, it is not immediately obvious which other simulators should receive a message notifying them of this update. The simulators modeling vehicles that can "see" the moving vehicle should receive

notification, however, the federate modeling the moving vehicle does not know which other federates are modeling these vehicles. Data distribution is typically accomplished by RTI software that matches the interests of each federate with information concerning the messages that are produced. In the High Level Architecture, for example, the data distribution management (DDM) services match data publishers with subscribers, and route messages to federates accordingly. Computations and communication are required to determine who is to receive what messages, and to maintain information necessary to determine receivers, translating into power consumption.

3 DEFINITIONS AND TERMS

Energy and power are two related, but distinct quantities. Energy is commonly defined as "the capacity for doing work" (Encyclopedia Britanica 2000), and here refers to the energy expended by the computing system to execute a distributed simulation. A joule corresponds to the energy required to move one coulomb of electric charge through an electric potential of one volt. Power refers to the amount of energy consumed per unit of time, with one watt referring to the expenditure of one joule of energy per second. Minimizing energy usage and power consumption are not the same thing (Unsal 2008). For example, decreasing the clock rate of the processor can lead to less power consumption. However, this will usually lead to longer execution times and can increase the total amount of energy needed to complete the computation. Energy or power may be much more important depending on the context in which the simulation is operating. Here, we are concerned with both power and energy consumption, but use the term "power" whenever the distinction between the two is not important.

For computing platforms operating on batteries energy consumption is usually the principle concern. Batteries convert chemical energy from materials within the fuel cell into electricity. A battery has a fixed energy capacity that is defined as the amount of electrical charge the battery can deliver at its rated voltage, measured in amp-hours (Wikipedia 2017). Thus, the amount of energy consumed by a computation directly impacts the battery's lifetime. Reducing power consumption is not helpful in addressing battery lifetime or size concerns if it does not result in a reduction of the total amount of energy consumed by the computation.

Concerns regarding heat dissipation and electric power bills suggest that power may be a greater concern for parallel simulations rather than energy, per se. Certainly this is true if the objective is to meet a power cap constraint. Very often power is viewed as the main metric for parallel simulations, and energy for distributed simulations designed to prolong battery life. However, this is not always the case. For example, an area of increasing interest is micro-cluster servers composed of closely coupled power-efficient processors, the same processors used in cellular phones, for instance, operating in energy constrained mobile platforms. As such, energy consumption for parallel simulation codes executing on these platforms may be of greater interest than power consumption. Similarly, in distributed simulations where part or all of the simulation executes on back-end cloud computing platforms or the concern is heat generated by the mobile device, power consumption may be of great concern.

Power-aware and *energy-aware* systems are those where power or energy consumption is a principal design consideration. For example, power-aware systems may utilize techniques to change the system's behavior based on the amount of power being consumed. Energy-aware systems may modify the operation of the system based on the amount of energy remaining in batteries, e.g., reducing data sampling rates or using lower precision computations. Battery operated devices are *energy-constrained* systems because they operate with a finite amount of available energy; thus a design goal might be to minimize the amount of energy utilized by the computation as a whole, subject to certain execution time and accuracy constraints. On the other hand, in *power-constrained* systems such as supercomputers and data centers the amount of available energy is effectively unlimited, but a design goal may be to minimize the amount of time required to complete the computation given a certain maximum level of power consumption, or alternatively to minimize power consumption, subject to certain execution time

constraints. In real-time systems a common goal is to minimize energy consumption while ensuring that certain deadlines are met by the computation.

Finally, it should be noted that minimizing execution time does not necessarily result in minimal energy consumption, although the two are often closely correlated. Energy consumption is affected by many factors including the operation of the memory system, the number and complexity of computations performed by arithmetic circuits, and importantly, the amount of inter-processor communication that is required. A parallel or distributed computation that executes in a shorter amount of time may consume both more energy and more power if more communications are required.

4 POWER CONSUMPTION IN DISTRIBUTED SIMULATIONS

Power and energy consumption in distributed simulations must always consider other design goals such as execution time, meeting deadlines, throughput, model accuracy, and/or precision. This suggest taking an holistic view of the system as a whole. In order to characterize different approaches to realizing power-efficient distributed simulations, we use the framework depicted in Figure 1. This framework is not unlike that described in (Benini and G. De Michela 2000), but has been adapted to apply to distributed simulations. The framework has two dimensions. The vertical axis corresponds to a view of the hardware/software stack used to implement the distributed simulation system. Specifically, we differentiate between (1) the *simulation model layer* where the application-specific simulation program is defined, (2) the *simulation engine layer* that includes distributed simulation middleware, and (3) the *system layer* that includes the operating system and hardware upon which the simulation engine and simulation model execute. The second, horizontal axis differentiates between the power consumed for (1) computation, (2) memory and storage, and (3) communications. Different techniques defined in the software stack will impact power consumption in different parts of the system.

The simulation model layer includes the representation of the state of the system being modeled and the code for transforming this state from one time instant to the next. It includes those portions of the distributed simulation that are specific to the simulation application. Several design decisions in creating the simulation model can have a large impact on power consumption. Perhaps most importantly, the degree of detail at which the system is modeled in space (e.g., the level of aggregation) and time (e.g., the time step size) will impact the amount of memory required and memory access patterns as well as the amount of computation required to update the system state. The model detail and abstractions that are used will also impact the amount and frequency with which data must be communicated among the processes making up the distributed simulation. Decisions such as the precision at which data is represented and algorithms used to transform the state of the simulation model will similarly impact energy consumption. Techniques such as dead reckoning (DR) may be used to reduce the amount of communication required, and thus the amount of energy expended for communications, at the cost of increased computation to execute the DR models. Design of the distributed simulation model will require

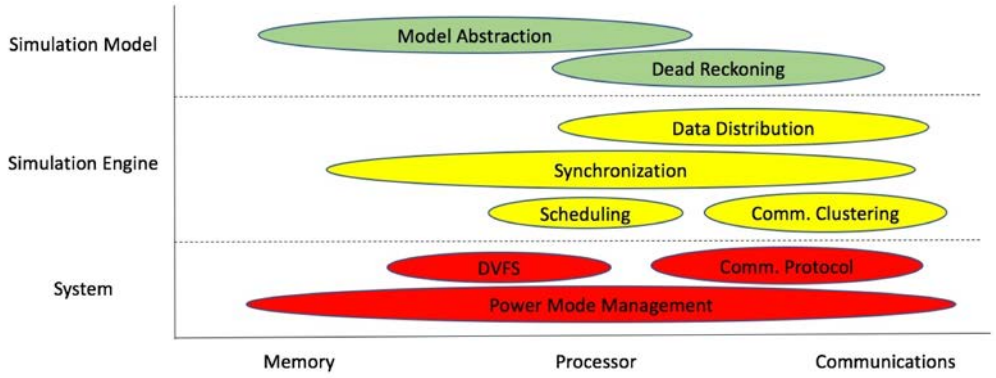

Figure 1. Framework of techniques for creating power efficient distributed simulations.

one to determine the minimal level of detail required to meet the accuracy and precision objectives of the simulation while also meeting runtime performance, memory, and power consumption constraints.

The simulation engine layer includes distributed simulation software that does not depend on the system being modeled. It includes the synchronization algorithm as well as other functions such as event list processing and data distribution. Design decisions concerning the implementation of these services can have a significant impact on power consumption. Different approaches to implementing data distribution and logical process scheduling will impact power. Clustering messages for inter-processor communications can also be used to reduce power consumption.

The systems layer includes the underlying hardware and operating system over which the distributed simulation executes. Low power operating systems focus on techniques such as light-weight implementation of essential services to reduce power consumption while meeting performance requirements and/or task completion deadlines (Cho et al. 2011; Quan and Hu 2001; Saewong and Rajkumar 2003). Many communication protocols are designed or optimized for low power operation. *Power mode management* techniques involve exploitation of different modes of operation for processors, memory, storage, and communication circuits (Bhatti et al. 2010; Hoeller et al. 2006; Niu and Quan 2004). For example, such components can often be disabled or switched to power saving states. The simulation computation can be mapped to a minimal number of processors necessary to meet delay and throughput needs, and the remaining cores can be powered down to reduce power consumption. Similarly, communications circuits can be powered down between communications. These techniques come at a cost, of course, e.g., in terms of increased execution time or longer communication latency. Many processors provide dynamic voltage and frequency scaling (DVFS) where the processor voltage and clock frequency can be reduced to trade off power consumption and performance (Freeh et al. 2007; Ge et al. 2005; Hua and Qu 2003).

There is a substantial literature in power aware design of embedded systems and a growing literature related to high performance computing. Most of this work focuses on the systems layer described above. Below we describe work specifically focused on power efficient distributed simulation. This includes both aspects unique to distributed simulations such as synchronization algorithms, as well as methods involving the application of general techniques such as power mode management to distributed simulations. The sections that follow discuss techniques within each of the three layers shown in Figure 1.

5 HARDWARE: COMPUTATION, STORAGE, AND COMMUNICATION

At the hardware level, power consumption for CMOS circuits is broken down into three main components – power resulting from the current needed to charge capacitive loads as signals change, short circuit current that occurs momentarily when a CMOS circuit switches, and power due to leakage current as indicated by the three terms below (Mudge 2001):

$$P = ACV^2f + \tau AVI_{short}f + VI_{leak}$$

Here P indicates power consumed by the circuit, A indicates the activity of the circuit (not all circuits switch on each clock), C is the total capacitive load on the circuit, V is the supply voltage, f is the clock frequency, τ indicates the time duration when the short circuit current flows, and I_{short} and I_{leak} are the short circuit and leakage current, respectively. The first two terms form the *dynamic* power consumption component and result from the operation and switching of active circuits. The third forms a *static* power consumption component that results from the circuit simply being powered on. In CMOS circuits dynamic power consumption usually dominates, and for the voltages typically used the clock frequency is proportional to voltage. This suggests that power is roughly proportional to the clock frequency cubed.

5.1 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) can be used to reduce dynamic power consumption. From the above discussion it is apparent that reducing the supply voltage V and the clock

frequency *f* is a way to significantly reduce the dynamic power consumption of the circuit. These techniques will reduce performance, however, leading to slower running programs. To a first order approximation, processor speed is proportional to *f*; more precisely execution time is equal to the number of machine instructions executed times the average number of clock cycles per instruction (CPI) times the clock cycle time, which is the reciprocal of *f*. Dynamic power consumption *P* as a function of processor speed *s* is generally assumed to be $P(s) = s^{\alpha}$ for some $\alpha > 1$; as suggested by the above discussion, α is often assumed to equal 3 leading to the well-known cube-root rule, i.e., $P(s)=s^3$, or the speed is proportional to approximately the cubed root of power (Brooks et al. 2000). For example, reducing power consumption by 50% results in the processor speed declining by approximately 20%, or execution time increasing by approximately 25%.

If the goal is to minimize power consumption with no additional constraints, then it is clear the best approach is to simply use the lowest frequency/voltage setting offered by the hardware, assuming static power consumption is negligible. A more interesting question is to select frequency/voltage settings to minimize power while meeting certain runtime performance goals.

Another hardware approach to reducing power consumption is to use unconventional processing cores, e.g., GPUs, FPGA, or specialized hardware, which are generally more power efficient than general purpose cores. A third approach discussed in greater detail next is to use power management modes. This means powering down certain elements of the system to reduce power consumption, e.g., processing cores or communication circuits. This technique reduces both static and dynamic power consumption.

5.2 Power Mode Management

DVFS effectively reduces power consumption by reducing the frequency and voltage of the CPU at a cost of slower execution speed. Taken to the extreme, one could simply power down one or more processing cores to reduce power consumption. This approach is not limited to the processor. Power can be saved by powering down communication circuits when they are not needed. Hardware for power efficient execution often provides several modes of operation to support techniques such as this. Manipulating the operating mode of circuits to reduce power consumption while still maintaining acceptable performance is referred to as *power mode management*.

Power mode management can be used, for instance to manage the execution of parallel simulations to operate within a power cap. Effective use of power mode management techniques requires an understanding of the relationship between execution time and power in order to achieve the best possible performance within the given power constraints. Power aware speed up (Ge and Cameron 2007) is defined as the ratio of the execution time of the system at the specified power level divided by the execution time at the lowest possible power level. An application with higher power-aware speedup is more sensitive to change in power levels.

One study of a parallel discrete event simulation of a telecommunication network is reported in (Fujimoto et al. 2017). An MPI-based parallel simulation of the NS3 network simulation package was used in this study. By disabling CPU cores the power consumption can be reduced at a cost of increased execution time. In addition the board used in this study included a Low Power Core in addition to 4 general CPU cores. Two boards connected to a wired private LAN were used for the experiments. A series of experiments were completed to evaluate the impact of disabling cores on performance and power consumption.

These experiments indicated that utilizing a 5^{th} core in this system, which require use of a second board, resulted in increased power consumption, but did not produce a corresponding reduction in execution time. As the number of cores was increased, performance only improved with the introduction of the 7^{th} core, suggesting using 5 or 6 cores is not advisable from a power perspective. While these experiments are specific to a particular hardware platform and application, the measurements highlight the fact that one may make different choices in configuring their application if power consumption is considered as a cost. Further, these experiments indicated that the lower power core used approximately

82% as much power as the high power (general) core, but the simulation ran 3.5 times slower, significantly more than that suggested by the cube-root rule discussed earlier.

A system can conserve power and energy by restricting power consumed by other elements of the system as well. One such element of interest is the network. Networks are generally over-provisioned both in terms of bandwidth and availability. In addition to the off-board power consumption of the network, the on board network/LAN card could also be powered down or turned off to conserve power and energy consumed by the system. This approach is best suited for synchronous PDES programs that utilize global synchronization points, e.g., YAWNS, and perform communications in bursts rather than continuously throughout the computation. Increasing the time between communications to 16 seconds resulted in utilizing approximately 57% of the power compared to using the minimum sleep period of 20 milliseconds (Fujimoto et al. 2017). However, one constraint in using this technique is the minimum period between communications. This is introduced by the hardware and/or software requirements of transitioning from one state to another. Some delay is required for the system to connect back to the LAN network. The total delay in this study was found to be on the order of 5 seconds.

6 SIMULATION MODEL

The simulation model layer includes the code specific to the application. Task graphs, widely used both in the distributed simulation and power consumption literature, are a useful representation of the distributed simulation computation.

6.1 Task Graphs

The computation within each logical process (LP) of a distributed simulation is a sequence of timestamped event computations, where each computation may (1) modify one or more state variables, and/or (2) schedule new events. Events within each LP must be processed in timestamp order.

Task graphs are a natural approach to represent distributed simulations and have long been used for this purpose. Each event computation is referred to as a task, and is represented as a node in the graph. Let $E_{i,j}$ represent the *j*th event in LP *i* where events within an LP are ordered according to timestamp. Arcs represent precedence relationships between tasks, i.e., an arc from event $E_{i,r}$ to $E_{j,s}$ indicates that the computation for $E_{i,r}$ must be completed before the computation for $E_{j,s}$ can begin.

In a discrete event simulation a precedence relationship exists if:

- $E_{i,r} \rightarrow E_{i,r+1}$: a precedence relationship exists between successive events within the same LP, or
- $E_{i,r} \rightarrow E_{j,s}$ if event computation $E_{i,r}$ resulted in generating event $E_{j,s}$.

The precedence relationship is transitive. Figure 2(a) shows a task graph for a discrete event simulation with each box representing an event computation; the event's location on the horizontal and vertical axes indicate the event timestamp and the LP processing the event, respectively. For example, these events might correspond to arrivals of a vehicle at an intersection in a traffic simulation or a traffic signal change. Arcs between LPs indicate events scheduled by one LP to be processed by another. The values in each box indicate the event computation subscripts $E_{i,j}$ as defined above.

In a time-stepped simulation all of the event computations for one time step must be completed before the event computations in the next time step can begin. Figure 2(b) shows the precedence graph for a time-stepped simulation.

Precedence graphs can be used to determine the parallelism and minimum execution time of distributed simulations (Berry and Jefferson 1985). They have also been used extensively to estimate power consumption of embedded computations, e.g., see (Brown et al. 1997; Dave et al. 1999; Kirovski and Potkonjak 1997). This is accomplished by mapping each task to the hardware responsible for completing the task and augmenting the task graph to specify the amount of energy required to complete the task. As such, task graphs are a useful way to model power consumption of distributed simulations.

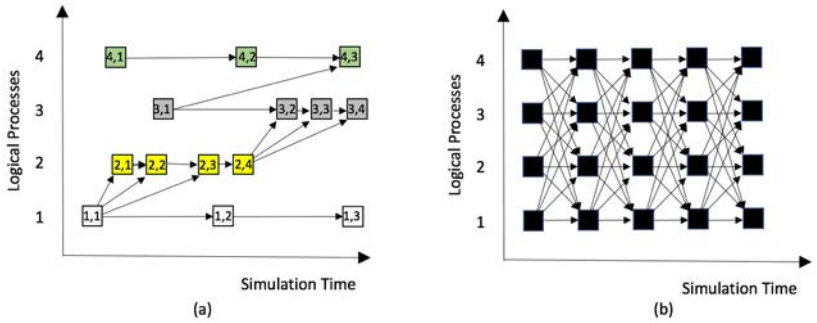

Figure 2. Task graphs for distributed simulations. (a) event-driven simulation. (b) time-driven simulation.

For example, in Figure 2(a) if we assume each event computation requires one unit of time and one unit of energy, it is easy to see that this task graph will require at least 8 units of time and 14 units of energy. Later, we will show how the task graph model can be used to analyze power consumption.

Note that the above model does not consider the energy consumed for communications or other operations such as managing event lists or other data structures. These are accounted for in the simulation engine, as discussed later.

6.2 Model Abstraction

One of the principle means of controlling the amount of power consumed in the simulation model layer is the abstraction and level of detail used by the model. The impact of the modeling approach on the amount of computation and memory required is clear. Just as model resolution is a way of managing execution time, it also presents a means for managing the power consumed by the simulation.

For example, one empirical study compared the power consumed for two different models of vehicle traffic: a time-stepped cellular automata (CA) model and a queueing model (Neal et al. 2016). In this study the CA simulation models the micro level dynamics of traffic flow behavior (Nagel and Schreckenberg 1992). Each road segment is divided into cells. The state of each cell is either occupied or empty to indicate if a vehicle currently resides within the cell. The simulation executes in a time-stepped fashion where the state of each cell is updated each time step in accordance with rules for vehicle movement. In the queuing model simulation traffic lanes are represented using queues that hold vehicles occupying a lane. The model state includes information concerning vehicles, intersections, and road segments. An event-driven execution paradigm is used with the event list implemented using a binary heap. Event handlers implement new vehicle arrivals and vehicle departures, as well as events modeling operations within each intersection. The latter includes events for vehicles arriving at, entering, crossing and departing from the intersection. Other events model traffic signal change events.

The models were configured to simulate a road segment in midtown Atlanta, and driven by measurement data of the same area. The cellular automata simulation required significantly more energy than the queueing model. This was attributed to the fact that the CA model required more computation to update each cell each time step. This more than compensated for the fact that the CA utilized simpler data structures. While these measurements pertain to a specific implementation of the simulation models, this example illustrates how the abstraction used in the model impacts power consumption and also highlights some of the issues one might consider during the development of power efficient simulation models.

6.3 Dead Reckoning

It is sometimes possible to reduce communication in a distributed simulation at the expense of increased computation. One example is the use of dead reckoning algorithms. Dead reckoning is a technique developed for real-time distributed simulations to reduce the amount of communications that are required (Lin and Schab 1994; Miller and Thorpe 1995; Saunders 1991). A local dead reckoning model computes the estimated position of entities, typically vehicles, modeled on other computers based on information

reported previously by the vehicle, e.g., its position, direction of travel, speed and acceleration. When the current position of the remote entity is required, the dead reckoning model is used to estimate its location.

The dead reckoning model's prediction will become inaccurate if the vehicle's motion deviates from that last reported, e.g., if it turns to a new direction or begins to accelerate or decelerate. To limit the resulting error, the processor simulating the vehicle also monitors the dead reckoning model, and if the difference between the dead reckoned position deviates from the actual position by more than a defined threshold, an update message is sent to update the remote dead reckoning models.

Dead reckoning is an example of a technique that reduces inter-processor communications at the expense of increased computation. Because communication is relatively expensive in terms of power, it provides a technique to reduce power consumed by the distributed simulation. Further, if the amount of required communication can be reduced so that there is a significantly long delay between communications, the communications circuits can be switched off further reducing energy consumption. This approach is explored in (Shi et al. 2003) where an adaptive dead reckoning algorithm is proposed. More broadly, techniques such as dead reckoning provide a means for trading off computation and communications to reduce power consumption.

7 SIMULATION ENGINE

The simulation engine includes functionality such as LP scheduling, synchronization and data distribution to implement services required by the distributed simulation. We next highlight how LP scheduling can impact power consumption and discuss the power cost of conservative and optimistic synchronization as well as approaches to data distribution and communications.

7.1 Logical Process Scheduling

The mapping of the distributed simulation computation to processors can have a significant impact on power consumption. To illustrate this point, consider a distributed simulation represented as a task graph, as discussed earlier. Specifically, consider the discrete event simulation shown in Figure 2(a). Assume each event consumes one unit of energy and requires one unit of time to complete.

Two executions of this task graph over time are shown in Figure 3. In Figure 3(a) each LP is mapped to a separate processor, and it is assumed that each event computation is performed as soon as its precedence constraints have been satisfied. For example, events $E_{1,1}$, $E_{3,1}$, and $E_{4,1}$ are processed in the first unit of wall clock time, $E_{1,2}$, $E_{2,1}$, and $E_{4,2}$ in the second, etc. The execution is completed in 8 time units, equal to the critical path execution time. An alternate execution of the same computation is shown in Figure 3(b) where LPs 1 and 2 are mapped to processor 1 and LPs 3 and 4 are mapped to processor 2. Arcs showing the scheduling of events are omitted to simplify the diagram. This execution also achieves the minimum, critical path execution time. Here, we assume an idealized conservative synchronization algorithm that can identify when each event can be processed without violating precedence constraints.

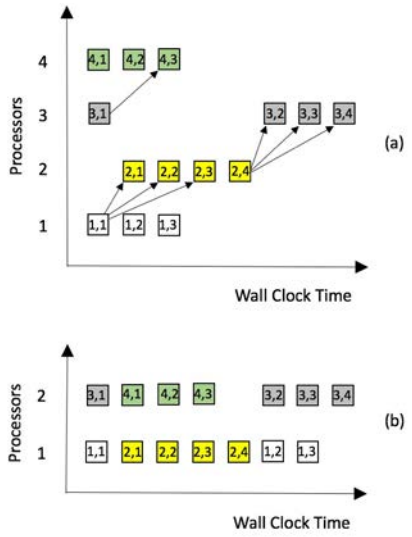

Figure 3. Energy and power consumption for two mappings of LPs to processors.

In a distributed simulation execution where battery life is the primary concern, we see that both executions consume a total of 14 units of energy, although it is noted that the two-processor execution requires less communication between processors, which will result in less energy consumption.

In a parallel execution where the maximum amount of power that can be utilized by the computation is capped, an analysis of the maximum power consumed by the computation may be useful. The maximum power using 4 processors is 3 because the maximum number of events that are processed concurrently is 3. The maximum power consumption in the two-processor case is 2, suggesting that this mapping can reduce power without sacrificing execution time. While this is clearly a simplified representation of the computation, this example does demonstrate that the approach to mapping LPs to processors can impact the maximum power consumption of the computation.

This simple analysis suggests that the mapping of processors to computing resources requires further consideration in evaluating power consumption for distributed simulations. This is an area that requires further research. Scheduling algorithms used in other areas, e.g., embedded systems, may be applicable here. A challenge for distributed simulations is, of course, the precedence relationships are in general not known prior to the computation, however, the task graphs provide a means to analyze the execution to derive first order bounds on power consumption.

7.2 Conservative Synchronization

As discussed earlier, distributed simulations require a synchronization algorithm to ensure the distributed execution yields the same results as a sequential execution. Because synchronization algorithms require a significant amount of inter-processor communication, one might expect a significant amount of energy will be required to ensure proper synchronization.

One study compared the power consumed by the asynchronous Chandy/Misra/Bryant (CMB) null message algorithm (Bryant 1977; Chandy and Misra 1979) and the synchronous YAWNS (Nicol 1993) algorithm. The energy cost of these conservative synchronization algorithms has been studied in (Biswas and Fujimoto 2016a, b; Fujimoto and Biswas 2015) as the lookahead in the distributed simulation was varied. One conclusion from this study was that CMB and YAWNS exhibit different behaviors with respect to energy consumption. In CMB the energy consumed steadily decreased as the lookahead was increased, a behavior attributed to creating cycles of null messages, a phenomenon known as lookahead creep. On the other hand, YAWNS energy consumption remains at a relatively constant level for small to moderate lookahead values, but then steadily decreased with increased lookahead at relatively high lookahead values. This was attributed to the ability of YAWNS to immediately jump to the timestamp of the next unprocessed event if the simulation had low lookahead, circumventing the lookahead creep problem. When the lookahead is significantly larger than the simulation time between event, to a first order approximation YAWNS will behave more like a time stepped simulation, and process all events within a lookahead sized time step. In this mode of operation increasing the lookahead will result in an increase in the number of events that are processed before the next global synchronization operation, resulting in a reduction in energy proportional to the lookahead.

Overall, these studies indicated that the synchronization algorithm can consume a significant amount of energy in executing the distributed simulation. Because the computations required to implement synchronization algorithms is usually minimal, energy consumption is driven largely by the amount of communication that is required. As such, the main challenge in creating power-efficient conservative synchronization algorithms is to minimize the number of messages without unnecessarily blocking LPs. Reducing the number of synchronization messages tends to also improve the performance of conservative synchronization algorithms, however, it remains to be seen if selecting the synchronization algorithm to minimize execution time also leads to one that minimizes energy.

7.3 Optimistic Synchronization

The power overhead associated with conservative synchronization largely concerns the messages that must be sent to ensure proper synchronization. In optimistic algorithms, the power overhead takes on a different form. There are several sources of power consumption. These include the power expended to (1) execute events that are later rolled back, (2) perform state saving tasks, (3) perform rollback

computations, (4) send, receive, and process anti-messages, and (5) compute GVT and other operations associated with fossil collection that are not present in a sequential execution.

There has been only a limited amount of work evaluating the power consumption of optimistic algorithms. One experiment found that the main overhead with respect to power consumption arose from processing events that were later rolled back (Fujimoto et al. 2017). These measurements also reported that the percentage of energy required for synchronization can be significant, as much as 40% in these measurements. These results suggest that the power overhead for Time Warp can be significant, and additional work to understand and optimize power in optimistic simulations is needed.

7.4 Power Consumption for Communications

Communication is a significant source of power consumption in distributed simulations. Sending messages generally requires much more power than receiving. One study reported that sending a stream of data from a cellular phone using 802.11 required approximately five times as much power as receiving (Fujimoto et al. 2017).

One approach to reducing energy consumption is to aggregate messages in the data stream. If the simulation must send a stream of update messages, one could aggregate several messages into a single message, and send one larger message rather than a sequence of smaller messages. This approach, termed message aggregation, is commonly used in distributed systems in order to reduce communication overheads. Message aggregation comes at the cost of increasing latency as some messages must be held at the sender in a buffer while the data is being accumulated. A set of experiments was conducted to consider the impact of aggregation on energy consumption. One study of this issue showed that the power required to send a stream of messages could be reduced more than six-fold using message aggregation, though savings were reduced significantly if the message had to be partitioned across multiple packets in the communication network (Fujimoto et al. 2017).

7.5 Dynamic Data Distribution Management

Data Distribution Management (DDM) is a set of services defined in the High Level Architecture (HLA) standard (IEEE Std 1516.3-2010 2010) to disseminate information among the federates (simulators), based on dynamically changing information such as a vehicle's location. DDM is based on an abstraction called the routing space, that is simply an N-dimensional coordinate system. Each message that is sent is associated with a rectangular update region. Each federate specifies a rectangular subscription region that indicates the portion of the routing space of interest to the federate. If the update region associated with a message overlaps a federate's subscription region, the federate should receive a copy of the message.

Several approaches to implementing the DDM services have been proposed. Perhaps the most direct is the region-based approach. A multicast group is defined for each publication region. Each federate simply joins those groups that correspond to publication regions that overlap with its subscription regions (Boukerche and Dzermajko 2001). A matching computation must be performed, often centrally, to determine overlaps between subscription and publication regions, in order to populate the multicast groups. This, of course, requires a certain amount of power to complete the matching computation. Whenever a publication (subscription) region changes, the new region must be compared against all other subscription (publication) regions to determine overlaps with the new region. By contrast, the grid-based implementation partitions the routing space into grid cells, and assigns a multicast group to each cell, circumventing most of the matching computations, but at the cost of sending additional, unnecessary messages to federates, thereby wasting power. Other hybrid approaches have been proposed, e.g., see (Boucherche and Roy 2002; Tan et al. 2000).

Trade-offs between computation and communications in implementing several DDM approaches are described in (Fujimoto et al. 2017). It is clear that utilizing a grid structure greatly reduces the energy needed for computation, but at a cost of increased energy for communications. This energy cost can be significant. Hybrid and dynamic grid schemes were also found to reduce power.

8 CONCLUDING REMARKS

Power is now a very important issue in many areas of computing, but has seen only limited attention by the modeling and simulation research community. Analysis and development of new techniques to improve power efficiency in distributed simulations is an open field, with many unresolved questions and problems. The space of techniques can be viewed in the context of the software stack, encompassing the simulation application, simulation engine or middleware, and the underlying system. Approaches impact power consumption in the processor, memory and storage system, and communications network.

The first step in optimizing power in distributed simulations is to understand the relationship among modeling approaches, synchronization and data distribution algorithms, hardware techniques, simulation accuracy and reliability, and power consumption. Fundamental understandings will enable the development and evaluation of new techniques to reduce power, subject to traditional modeling objectives. In some cases, approaches can build upon other work in parallel and distributed computing in general. Other efforts will need to focus on aspects specific to distributed simulations.

Work in power consumption in parallel and distributed simulations is in its infancy. There is very little work to date concerning the development of new energy efficient approaches in key areas such as synchronization and data distribution. As such, power-efficient parallel and distributed simulation is an area with many open research questions.

ACKNOWLEDGMENTS

Funding was provided by NSF/AFOSR Grant 1462503 and AFOSR grant FA9550-17-1-022.

REFERENCES

Benini, L., and G. De Michela. 2000. "System-Level Power Optimization: Techniques and Tools." *ACM Transactions on Design Automation of Electronic Systems* 5 (2):115-192.

Berry, O., and D. Jefferson. 1985. "Critical Path Analysis of Distributed Simulation." In *Proceedings of the SCS Conference on Distributed Simulation*, 57-60.

Bhatti, K., C. Belleudy, and M. Auguin. 2010. "Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of Dpm and Dvfs Policies." In *International Conference on Embedded and Ubiquitous Computing*, 184–191. IEEE.

Biswas, A., and R. M. Fujimoto. 2016a. "Energy Consumption of Synchronization Algorithms in Distributed Simulations." *Journal of Simulation*.

Biswas, A., and R. M. Fujimoto. 2016b. "Profiling Energy Consumption in Distributed Simulation." In *Principles of Advanced Discrete Simulation.*

Boucherche, A., and A. J. Roy. 2002. "Dynamic Grid-Based Approach to Data Distribution Management." *Journal of Parallel and Distributed Computing* 62:366-392.

Boukerche, A., and C. Dzermajko. 2001. Performance Comparison of Data Distribution Management Strategies. Proceedings of the 5th IEEE International Workshop on Distributed Simulation and Real-Time Applications.

Brooks, D. M., P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. 2000. "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors." *IEEE Micro* 20 (6):26–44.

Brown, J. J., D. Z. Chen, G. W. Greenwood, X. Hu, and R. W. Taylor. 1997. "Scheduling for Power Reduction in a Real-Time System." In *Proceedings of the International Symposium on Low Power Electronics and Design*, 84–87. New York, NY: ACM Press.

Bryant, R. E. 1977. "Simulation of Packet Communication Architecture Computer Systems." M.S. thesis, MIT-LCS-TR-188, Computer Science Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

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.

Cho, K.-M., C.-H. Liang, J.-Y. Huang, and C.-S. Yang. 2011. "Design and Implementation of a General Purpose Power-Saving Scheduling Algorithm for Embedded Systems." In *IEEE International Conference on Signal Processing, Communications and Computing*, 1–5. IEEE.

Darema, F. 2004. "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements." In *International Conference on Computational Science*, 662-669. Springer.

Dave, B., G. Lakshminarayana, and N. Jha. 1999. "Cosyn: Hardware-Software Co-Synthesis for Heterogeneous Distributed Embedded Systems." *IEEE Transactions on Very Large Scale Integration Systems* 7 (1).

Encyclopedia Britanica. 2000. "Energy (Physics)." Accessed Accessed March 26, 2017. https://www.britannica.com/science/energy.

Freeh, V. W., D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal. 2007. "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications." *IEEE Trans. Parallel Distrib. Syst.* 18 (6):835--848.

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

Fujimoto, R., M. Hunter, J. Sirichoke, M. Palekar, H.-K. Kim, and W. Suh. 2007. "Ad Hoc Distributed Simulations." In *Principles of Advanced and Distributed Simulation*, 15-24. IEEE.

Fujimoto, R. M., and A. Biswas. 2015. "An Empirical Study of Energy Consumption in Distributed Simulations." In *IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*.

Fujimoto, R. M., M. Hunter, A. Biswas, M. Jackson, and S. Neal. 2017. "Power Efficient Distributed Simulation." In *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM.

Fujimoto, R. M., D. Lunceford, E. Page, and A. Uhrmacher (editors). 2002. "Grand Challenges in Modeling and Simulation." Technical Report 350, Schloss Dagstuhl, Seminar No. 02351.

Ge, R., and K. W. Cameron. 2007. "Power-Aware Speedup." In *IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007.*, IEEE.

Ge, R., X. Feng, and K. W. Cameron. 2005. "Performance-Constrained Distributed Dvs Scheduling for Scientific Applications on Power-Aware Clusters." In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 34--. IEEE Computer Society.

Hoeller, A., L. Wanner, and A. Fröhlich. 2006. "A Hierarchical Approach for Power Management on Mobile Embedded Systems." In *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, 265–274.

Hua, S., and G. Qu. 2003. "Approaching the Maximum Energy Saving on Embedded Systems with Multiple Voltages." In *IEEE/ACM International Conference on Computer-Aided Design*, 26.

Huck, S. 2011. "Measuring Processor Power." Intel Corporation.

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 1516.3-2010. 2010. *Ieee Standard for Modeling and Simulation (M&S) High Level Architecture (Hla) -- Interface 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.

Kamrani, F., and R. Ayani. 2007. Using on-Line Simulation for Adaptive Path Planning of Uavs. Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications.

Kirovski, D., and M. Potkonjak. 1997. "System-Level Synthesis of Low-Power Hard Real-Time Systems." In *Proceedings of the Annual Conference on Design Automation*, 697–702. New York, NY: ACM Press.

Lin, K.-C., and D. E. Schab. 1994. "The Performance Assessment of the Dead Reckoning Algorithms in Dis." *Simulation* 63 (5):318-325.

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.

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

Mudge, T. 2001. "Power: A First-Class Architectural Design Constraint." *IEEE Computer* 34 (4):52-58.

Nagel, K., and M. Schreckenberg. 1992. "A Cellular Automata Model for Freeway Traffic." *J. Physique I* 2:2221-2229.

Neal, S., R. M. Fujimoto, and M. Hunter. 2016. "Energy Consumption of Data Driven Traffic Simulations." In *Winter Simulation Conference.* edited by T. Roeder, P. Frazier, R. Szechtman, E. Zhou, T. Huschka, S. Chick, 1119-1130, Piscataway, New Jersey, IEEE.

Nicol, D. M. 1993. "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations." *Journal of the Association for Computing Machinery* 40 (2):304-333.

Niu, L., and G. Quan. 2004. "Reducing Both Dynamic and Leakage Energy Consumption for Hard Real-Time Systems." In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 140–148.

Quan, G., and X. Hu. 2001. "Energy Efficient Fixed- Priority Scheduling for Real-Time Systems on Variable Voltage Processors." In *Design Automation Conference*, 828–833.

Saewong, S., and R. Rajkumar. 2003. "Practical Voltage- Scaling for Fixed-Priority Rt-Systems." In *IEEE Real- Time and Embedded Technology and Applications Symposium*, 106–114.

Saunders, R. 1991. "Formal Expression of Dead Reckoning: Mathematical and Representation Recommendation." In *DIS Workshop*.

Shehabi, A., S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner. 2016. "United States Data Center Energy Usage Report." Technical Report No. LBNL-1005775, Lawrence Berkeley National Laboratory.

Shi, W., K. S. Perumalla, and R. M. Fujimoto. 2003. Power-Aware State Dissemination in Mobile Distributed Virtual Environments. Workshop on Parallel and Distributed Simulation, 2003/06/01.

Tan, G., Y. Zhang, and R. Ayani. 2000. A Hybrid Approach to Data Distribution Management. Proceedings of the 4th IEEE International Workshop on Distributed Simulation and Real-Time Applications.

Unsal, O. S. 2008. "System-Level Power-Aware Computing in Complex Real-Time and Multimedia Systems." Doctoral Dissertation, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst.

Wikipedia. 2017. "Battery (Electricity)." Accessed Accessed March 26, 2017. https://en.wikipedia.org/wiki/Battery_(electricity).

AUTHOR BIOGRAPHIES

RICHARD FUJIMOTO is a 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. He has been an active researcher in the parallel and distributed simulation field since 1985, and has authored or co-authored 3 books and over 250 technical papers on this subject including 7 award winning publications. He led the definition of the time management services for the High Level Architecture for modeling and simulation (IEEE standard 1516). He received the ACM Distinguished Contributions in Modeling and Simulation Award in 2013. His e-mail address is fujimoto@cc.gatech.edu.