Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

A FAST HYBRID TIME-SYNCHRONOUS/EVENT APPROACH TO PARALLEL DISCRETE EVENT SIMULATION OF QUEUING NETWORKS

Hyungwook Park
Paul A. Fishwick

Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611, U.S.A.

ABSTRACT

The trend in computing architectures has been toward multi-core central processing units (CPUs) and graphics processing units (GPUs). An affordable and highly parallelizable GPU is practical example of Single Instruction, Multiple Data (SIMD) architectures oriented toward *stream processing*. While the GPU architectures and languages are fairly easily employed for inherently time-synchronous based simulation models, it is less clear if or how one might employ them for queuing model simulation, which has an asynchronous behavior. We have derived a two-step process that allows SIMD-style simulation on queuing networks, by initially performing SIMD computation over a cluster and following this research with a GPU experiment. The two-step process simulates approximate time events synchronously and then reduces the error in output statistics by compensating for it based on error analysis trends. We present our findings to show that, while the outputs are approximate, one may obtain reasonably accurate summary statistics quickly.

1 INTRODUCTION

A typical type of discrete event model is a queuing model. Queuing models are constructed to simulate humanly engineered systems where jobs, parts, or people flow through a network of nodes (i.e., resources). The study of queuing models, their simulation, and analysis is one of the primary research topics studied within the discrete event simulation community (Law and Kelton 2006).

Queuing model simulation can be expensive in terms of time and resources in cases where the models are composed of multiple resource nodes and tokens that flow through the system. Therefore, there is a need to find ways to speed up queuing model simulations so that analyses can be obtained more quickly. Past approaches to speeding up queuing model simulation have used asynchronous message-passing with special emphasis on two approaches: the conservative and optimistic approaches (Fujimoto 2000). Both approaches have been used to synchronize the asynchronous logical processors (LPs), preserving causal relationships across LPs so that the results obtained are exactly the same as those produced by sequential simulation. Most studies of parallel simulation have been performed on Multiple Instruction, Multiple Data (MIMD) machines or related networks to execute the part of a simulation model or LPs. This approach could easily be employed with a queuing model simulation since the start of each execution needs not be explicitly synchronized with other LPs.

Recently, parallel simulation has extended its boundary from PCs and workstations to programmable hardware such as a GPU, and a Cell processor (Kumar and Radha 2007). The GPU has become an increasingly attractive option as it is ubiquitous and has enough computational power to substitute for the expensive clusters of workstations, at a relatively low cost (Owens et al. 2005). However, the GPU is SIMD-based hardware oriented toward stream processing. SIMD hardware is a relatively simple, inexpensive, and highly parallel architecture, but some applications cannot easily be run using SIMD due to its lack of programming flexibility.

Most proposed works for simulation using the GPU are compute-intensive models with coarse-grained events while the queuing models are too fine-grained to benefit from the GPU. Moreover, other studies (Perumalla 2006; Ayani and Berkman 1993) have shown that time-synchronous simulation models are well suited to SIMD-style simulation while the queuing model simulation usually has asynchronous behavior, and thus may be less suitable.

This paper presents a new method for asynchronous queuing network simulation on SIMD hardware. The issue with using SIMD is that event times need to be modified so that they could be synchronized. This process naturally leads to approximation errors in the summary statistics yielded from the simulation. In our experiments, the error may be found to be acceptable for particular modeled applications where the analyst is more concerned with speed and can tolerate relatively minor inaccuracy in summary statistics.

978-1-4244-2708-6/08/$25.00 ©2008 IEEE

In some cases, the error can be approximated and potentially corrected to yield more accurate statistics.

This paper is organized as follows. Section 2 describes related work. Section 3 presents our simulation methodology with a hybrid time synchronous/event algorithm. Section 4 shows our experimental results, including accuracy and performance. Section 5 analyzes the errors produced by the time interval. Section 6 concludes this approach and presents future work.

2 RELATED WORK

2.1 Simulation on SIMD Hardware

In the '90s, efforts were made to parallelize discrete event simulation using a SIMD approach given that with a balanced workload, SIMD has the potential to significantly speed up a simulation. The research performed in this area was focused on replication. The processors were used to parallelize the choice of parameter by implementing a standard clock algorithm (Vakili 1992; Patsis, Chen, and Larson 1997). Ayani and Berkman (1993) used SIMD for parallelizing simultaneous event execution but SIMD was determined to be a poor choice because of the uneven distribution of timed events. There is a need to fill the gap between asynchronous applications and synchronous machines so that the SIMD machine can be utilized for asynchronous applications (Shu and Wu 1995).

Recently, the computer graphics community has widely published on the use of the GPU for physical and geometric problem solving. The types of models used here have the property of being decomposable over a variable or parameter space, such as cellular automata (Gobron, Devillard, and Heit 2007) for discrete spaces and partial differential equations (PDEs) (Harris et al. 2003; Nyland, Harris, and Prins 2007) for continuous spaces. Queuing models, however, do not adhere to this property.

Perumalla (2006) indicates that selective individual execution such as discrete event simulation is extremely inefficient on the GPU. He performed the first discrete event simulation on a GPU by running a diffusion simulation. His hybrid algorithm combines the time-stepped and discrete event algorithm to use the GPU as the sole architecture in the simulation because the event scheduling method is assumed not to be compatible with GPUs. A minimum event time was chosen from the list of update times, and used as a time-step to synchronously update all elements on a grid. It was found that the GPU is well-suited to a time-stepped algorithm and a significant performance improvement is expected when the problem size is larger than an L2 cache size of the CPU.

Xu and Bagrodia (2007) proposed the discrete event simulation framework for network simulation. They used the GPU to distribute compute-intensive workloads, such as differential equations and least square estimations for high-fidelity network simulations. The two examples of discrete event simulation are coarse-grained as well as time-synchronous models. Perumalla's hybrid algorithm requires updating of the entire grid because selective updates of individual elements are not possible on a grid.

2.2 Relaxed Synchronization

Relaxed synchronization is one of the synchronization methods to improve the performance in parallel simulation at the cost of accuracy. Tolerant synchronization (Martini, Rümekasten, and Tölle 1997) and unsynchronized discrete event simulation (Rao et al. 1998) are examples of relaxed synchronization. State-matching is the most dominant problem in a time-parallel simulation (Fujimoto 2000) as with synchronization in a space-parallel simulation. If the initial and final states are not matched at the boundary of a time interval, re-computation of those time intervals degrades simulation performance. Approximation simulations (Wang and Abrams 1992; Kiesling and Pohl 2004) have been used to solve this problem with a loss of accuracy in order to improve the simulation performance.

Fujimoto (1999) proposed exploitation of temporal uncertainty, which introduces approximate time. Approximate time is a time interval for the execution time of the event rather than a precise timestamp. The precise timestamp can be relaxed into the time interval due to temporal uncertainty. When approximate time is used, the time intervals of events on the different LPs can be overlapped on the timeline at one common point. Although events on the different LPs have to wait for a synchronization signal with a conservative method when a precise timestamp is assigned, events can be executed concurrently if their time intervals overlap with each other. The performance is improved due to increased concurrency, but at the cost of accuracy in the results of simulation. Our approach differs from this method in that we do not assign a time interval to each event, and events are clustered at a time interval when they are extracted from the future event list (FEL). In addition, an approximate time is executed based on a MIMD scheme that partitions the simulation model whereas our approach is based on a SIMD scheme.

Our research differs from the previous related work in the following ways:

- The time interval is used to execute events concurrently for the purpose of reducing the number of idle processors on a SIMD machine during simulation.
- The simulation runtime is reduced at the expense of accuracy, but timestamp ordering is still preserved, contrary to other approximation studies.

- The simulation model is not partitioned into several LPs, so complicated synchronization methods are not required.

3 HYBRID TIME-SYNCHRONOUS/EVENT APPROACH

3.1 Methodology and Algorithm

We used a parallel simulation method based on a SIMD scheme so that events with the same timestamp value are executed concurrently. If there are two or more events with the same timestamp, they are clustered into a list, and each event on the list, to be executed concurrently, is sent to each processor. The algorithm combines the processors, or CPU and GPU, into a master-slave paradigm. One master processor, or CPU, works as the control unit, and several slave processors, or GPUs, execute the programmed codes or kernels.

The simulation begins with the extraction of the event with the lowest timestamp from an FEL in the master processor. Event extraction continues for as long as the next event has the same timestamp. All events with the same timestamp are created as a current event list (CEL) from the FEL. Each event in the CEL is sent to one of the slave processors. When the master processor assigns events to slave processors, dynamic mapping is used between the logical and physical processors. After execution on a slave processor, the results and timestamp increments for the next execution are returned to the master processor. Then, the next corresponding event is scheduled to the FEL with a timestamp increment. Until all the results are received from the slave processors, the master processor does not proceed to extract the next event for the purpose of synchronizing the parallel simulation.

However, it is improbable that several events will occur at a single point of simulated time. In this case, many slave processors will be idle, waiting for the end of the current execution on other slave processors. This makes the overall performance inefficient. If the length of the timestamp is further away from the precise timestamp, more events can be gathered into the CEL. To have more events occurring concurrently and reduce the load imbalance across the processors, we introduce a time interval instead of a precise time. The master processor extracts events from the FEL at the end of the time interval. Clustering events that occur within a time interval makes it possible for many more events to be executed at a single point of simulated time, which prevents the slave processors from being idle, and achieves more effective parallel processing.

Figure 1 illustrates a time-synchronous/event algorithm written in Java pseudo-code. A time-synchronous/event algorithm is a hybrid algorithm of discrete event simulation and time-stepped simulation. The main difference between the two types of discrete simulation is a time-advance algorithm. Our approach is similar to a time-stepped simulation in that we execute events at the end of the time interval to improve the degree of parallelism. However, time-stepped simulation can be inefficient if the state changes in the simulation model occur irregularly, or if event density is low at the time interval. The clock has to advance to the next time-step with idle processing time if there is no event at the current time-step, and this reduces the efficiency of the simulation.

```
public static void main()
   while (currentTime <= simulationTime)
      eventList = NextEventTI(interval);
      executes eventList;
   end while
end main

public eventlist NextEventTI(interval)
    eventTime = the lowest timestamp from FEL;
    currentStep = the smallest multiple of
       time interval greater than or equal to eventTime;
    while (eventTime <= currentStep)
       currentList += currentEvent;
    end while
    currentTime = currentStep;
    return currentList;
end NextEventTI
```

Figure 1: Hybrid time-synchronous/event algorithm

Our approach is based on discrete event simulation in that the clock advances by the next event, rather than the next time-step. When the *NextEventTI()* method is called, it checks the event with the lowest timestamp from the FEL. The *NextEventTI()* method extracts all events from the FEL at a time when their timestamp is less than, or equal to the *currentStep*. The *currentStep* is the smallest multiple of the time interval that is greater than or equal to the current event time. Extracted events are clustered into a CEL and executed concurrently. The time interval in our approach is used to execute events concurrently rather than to advance the clock. After executing events, the clock advances to the next lowest event time, and not to the next time-step.

However, if events are executed only at the end of the time interval, the results lose accuracy because each event has to be delayed in its execution compared to its original timestamp. Fortunately, we can approximate the error due to the stochastic nature of queues. For small and non-complex queuing networks, the analytic model can provide the statistics without running a simulation based on queuing theory, albeit with assumptions and approximations (Kleinrock 1975; Bolch et al. 2006). We use queuing theory

to estimate the total error rate. The time interval can be another parameter of the queuing model combined with two other parameters: arrival and service rate. With the use of the time interval, the error rate caused by the time interval is related to the arrival and service rates, and the amount of error depends on the values of these parameters. The relationships between the time interval and parameters are described in Section 4 and 5.

3.2 Timestamp Ordering

In parallel simulation, the purpose of synchronization is to process the events in timestamp order to get the same results as those of sequential simulation. Relaxed synchronization allows the timestamp ordering to be violated within certain limits, whereas our approach does not allow it. We need not synchronize the clocks between processors due to the global event list and clock. The master processor does not proceed in extracting the next event list until it receives the results from all the slave processors. The error caused by the time interval is different from the causal error because the timestamp ordering is preserved even though events are clustered at the end of the time interval. The synchronous step of the simulation preserves the executions of events in timestamp order, blocking the event extractions from the FEL before the current events finish scheduling the next events.

A *lookahead* can be used as a time interval. If a time interval is set to be less than the lookahead in the simulation model, all events in the current time interval are independent of each other. The timestamp of the new event, generated by the currently executed events, must be larger than the time of the current time-step because the minimum increment of the next event time is greater than a time interval. This guarantees that it is safe to process the events in the current time interval at the same time since no events can be scheduled before the currently executed events. In this case, the causal relationship between events, as well as the timestamp ordering, is preserved as synchronization in a conservative approach.

3.3 Open and Closed Queuing Networks

Queuing networks are classified into two types: open and closed (Bolch et al. 2006). Let a *token* denote any type of customer that requests service at the service facility. The main difference between these two types of queuing networks is that the open queuing network has new arrivals during simulation. The number of tokens in the open queuing network at an instant of time is always different due to the arrival and departure rates. The closed queuing network has a constant number of tokens during simulation since there are no new arrivals and departures. The error rate produced by the use of a time interval will be different between the two queuing networks since the number of tokens in the system affects the simulation results.

In the open queuing network, the arrival rate remains constant even if the events are only executed at the end of each time interval. A delayed execution time for each event, compared to its precise timestamp, decreases the departure rate from the queuing network, resulting in an increased number of tokens in the system. As the number of tokens increases, the waiting time also increases since the length of the queue at the service facility increases. In the closed queuing network, we only need to consider the arrival and departure rates between the service facilities because there is no entry from the outside. The delayed tokens arrive at the next service facility as late as the difference between their original timestamps and actual execution times. The length of the queue at the service facility remains unchanged by the time interval because all tokens in the system are delayed at the same rate.

4 EXPERIMENTAL RESULTS

4.1 Simulation Environment

We initially conducted experiments over clusters of workstations. The clusters used for the simulation are composed of 24 Sun workstations interconnected by a 100Mbps Ethernet. Each workstation is a Sun SPARC 1GHz machine with a running version 5.8 of the Solaris operating system with 512MB of main memory. The application was developed using *SimPack* (Fishwick 1992). SimPack is a simulation toolkit which supports the construction of various types of models and is executing the simulation. The results represented in this paper are the average value of five runs.

4.2 Simulation Models

We used two kinds of queuing network models, closed and open queuing networks to identify the difference between the two models when we ran a simulation using the time interval. We compared the results of the closed queuing network with those of the open queuing network first, and analyzed the accuracy of the closed queuing network.

4.2.1 Closed Queuing network

The first model is the queuing network of the toroidal topology based on PHOLD (Fujimoto 1990). The application is an example of the closed queuing network interconnected with the service facility. Each service facility is connected to its four neighbors. When the token arrives at the service facility, the service time is assigned to the token by the random number generator with exponential distribution. After being served by the service facility, the token moves to one of its four neighbors selected with uniform distribution.

Communication delay of the null message between master and slave processors was measured as less than 1 millisecond (*ms*), but it overwhelms ten microseconds (μs) of the computation speed for each event. We configured the computation granularity to 1 *ms*, adding the 1 *ms* delay method to the event processing routine. The values of various parameters can be important factor affecting accuracy and performance. We ran the simulation with several parameter settings, such as the time interval, message population, mean service time, and the number of service facilities, to see the effect of those parameters.

4.2.2 Open Queuing Network

We modified the closed queuing network (CQN) model (Bagrodia and Takai 2000) into the open queuing network, as shown in Figure 2. The original model consists of N linear queuing networks with one switch and k servers. We modified this model, adding new arrivals from the calling population and the branch at the end of the linear queuing network. A new token arrives at the system based on arrival rate λ from the calling population. The new arrival token is assigned to one of the linear queuing networks with uniform distribution. After being served at the last server in the linear queuing network, the token completes its job and exits the system with probability P_o, or is assigned to the switch with probability P_i. The switch forwards the token to one of the linear queuing networks with uniform distribution. The arrival and service times are determined by exponential distribution.

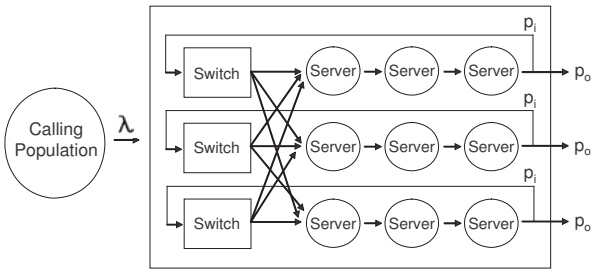


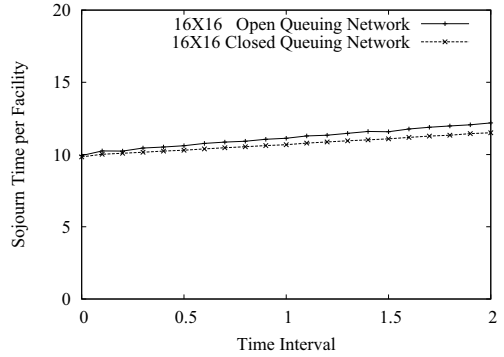
Figure 2: 3 (switches) × 3 (servers) open queuing network

4.3 Accuracy

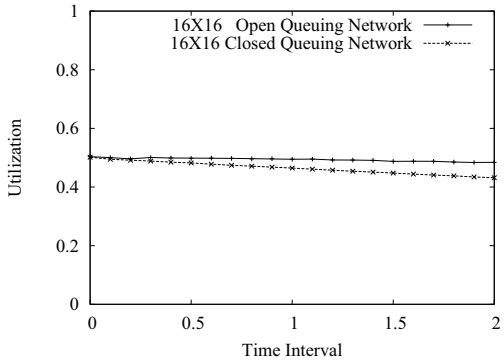
4.3.1 Open vs. Closed Queuing Network

The values of parameters and the number of service facilities for closed and open queuing networks are configured to get similar results when the time interval is set to zero. The results for various time intervals are compared with those of a time interval of zero to determine the accuracy. The mean service time of the facility is set to 5 with exponential distribution for both queuing networks. In the closed queuing network, the message population – the number of initially assigned tokens per service facility – is set to 1. In the open queuing network, the probability P_o is 0.25 and P_i is 0.75. We used the 16×16 topology as a basis for the experiment to determine the accuracy and performance.

Two summary statistics are presented to see the difference by using the time interval, as shown in Figure 3. Sojourn time is the average time for a token to stay per service facility including the wait time in the queue. Utilization represents the performance of the simulation model.



(a) Sojourn time



(b) Utilization

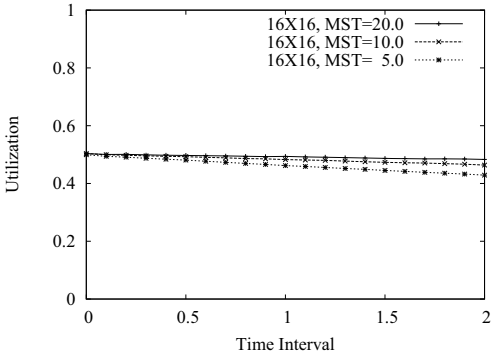
Figure 3: Summary statistics of queuing network simulations

In each subsequent plot, the time interval is on the horizontal axis. A time interval of zero indicates no error in accuracy. As the interval increases, the error also increases for the variable being measured on the vertical axis. Figure 3(a) shows the average sojourn time of open and closed queuing networks for the time interval. It takes much longer for a token to pass a facility in the open queuing network since the number of tokens grows in the open queuing network, compared to the closed queuing network as the time interval increases. Figure 3(b) shows the utilization for the time interval. Utilization of the closed queuing network drops since arrivals for each facility are delayed due to

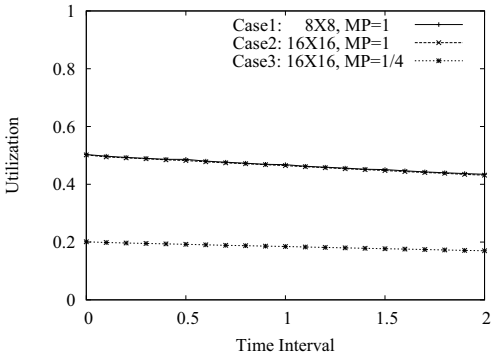
the time interval, but that of the open queuing network is almost constant since the increased number of tokens fills up possible idle time.

4.3.2 Effects of Parameter Settings on Accuracy

The closed queuing network was used for simulation to determine the effects of the parameter settings on accuracy. Figure 4(a) shows the utilization of the 16×16 toroidal queuing network, with variation in the mean service time for the time interval. As the mean service time increases, the ratio of the delayed time of a token at the service facility to the mean service time decreases. The error, therefore, decreases as the mean service time increases for the same time interval.



(a) Varying the service time



(b) Varying the MP and number of facilities

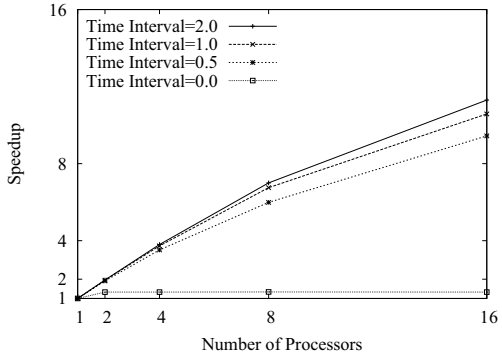
Figure 4: Statistics with varying parameter settings

Figure 4(b) shows the utilization with variation in the message population (MP) and the number of facilities for the three cases. Case1 and Case2 have the same density of tokens, and Case1 and Case3 have the same number of tokens. The same message population produces nearly the same error in the results, regardless of the number of facilities. We can say that the number of facilities has little impact on the error rate, as shown in Figure 4(b).

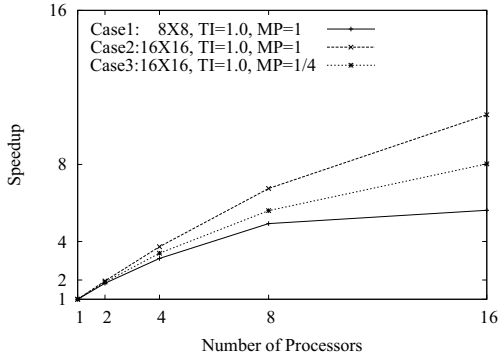
4.4 Performance

The performance is calculated by comparing the runtime of a parallel simulation with that of a sequential simulation. We can expect better performance as the time interval increases since many events are clustered at one time interval, however, a large time interval also introduces more errors in the results.

Figure 5(a) shows the improvement in the performance of the closed queuing network for the number of processors and the time interval, with the same values of parameters that were used in Figure 3. As expected, a larger time interval leads to better performance. However, performance was not improved even though the number of processors was increased from 2 to 16 when a SIMD parallel simulation scheme was used in the case where the time interval was set to zero. Many processors are idle since there are few events at a single point of simulated time. A large time interval does not always yield good performance. The performance improvement is related to both the number of events in one time interval and the load balance. Specific information for the simulation model is often needed to determine the level of acceptable accuracy loss and the desired speed improvement in the simulation.



(a) Varying the time interval



(b) Varying the MP and number of facilities

Figure 5: Performance improvement

Figure 5(b) shows the performance improvement for the message population and the number of facilities with

the same parameter settings, as shown in Figure 4(b). This graph also indicates that the speed of the simulation is heavily dependent upon the number of events during a time interval.

4.5 GPU Experiment

The GPU experiment was conducted on a Dell XPS 710. The XPS has an Intel Core 2 Extreme Quad processor with 2.66GHz and 3GB of main memory. Two Nvidia GeForce 8800 GTX GPUs (NVIDIA Corporation 2006) are deployed on the XPS. Each GPU has 768MB of memory with a memory bandwidth of 86.4GB/s. The CPU communicates with the GPU via PCI-Express with a maximum of 4GB/s in each direction. The GeForce 8800 GTX GPU is the first GPU model unifying vertex, geometry and fragment shaders into 128 individual stream processors (SPs). Each SP can process the instructions in SIMD fashion. The application was developed using the CUDA (NVIDIA Corporation 2007) with SimPack. CUDA is a unified hardware and software solution that allows the GeForce 8 series GPU to process kernels on a specified number of threads. The active events are selectively extracted from an event list and mapped into streams as an array. The functions for event execution are programmed into the kernel. This approach allows the CPU to process an event list and to create streams while the GPU executes streams in parallel.

The toroidal queuing network model was used for simulation with a mean service time of 5, a time interval of 1 and the message population of 1, with variations in the number of facilities. During the simulation, the GPU produces two random numbers for each active token; the service time at the current service facility by exponential distribution, and next service facility by uniform distribution. When the CPU calls the kernel and passes the streams of active tokens, threads on the GPU generate the results in parallel, and return them to the CPU. The CPU schedules the tokens using these results. No artificial delay method was used in the GPU experiments.

Figure 6 shows the performance improvement in the GPU experiments. The CPU-based simulation showed better performance in the 16×16 facilities because (1) the sequential execution time in one time interval on the CPU was not long enough compared to the data transfer time between the CPU and GPU (2) the number of events in one time interval was not enough to maximize the number of threads on the GPU. The GPU-based simulation outperforms the sequential simulation when (1) is satisfied, and the performance increases when (2) is satisfied. However, the performance was not good enough when we compare the results with other coarse-grained simulations. In the SIMD execution, some parts of codes are processed in sequence, such as the instruction fetch. The event processing method (e.g., the event insertion and extraction) performed in sequence represents over 95% of the overall simulation time while the event execution time (e.g., random number generation) is reduced by utilizing the GPU. If we can parallelize this part on the GPU, much better performance is expected since event processing method occupies the large portion of the overall simulation time in discrete event simulation.

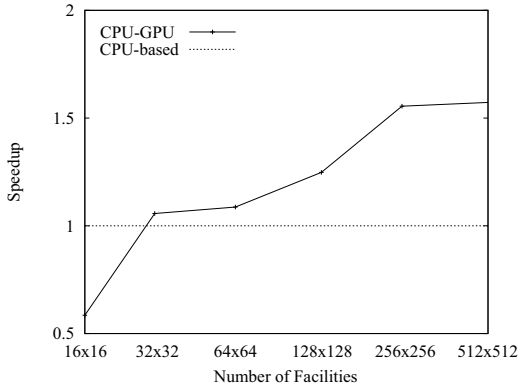


Figure 6: Performance improvement by GPU experiment

5 ERROR ANALYSIS

In this section, we explain how the error equation is derived and the error is corrected to improve the accuracy of the resulting statistics. The closed queuing network is used for these analyses because there are, relatively, more parameters to consider in the open queuing network, such as the arrival rate and routing probability.

When a token is clustered at the end of the time interval, the token is delayed by the amount of time between the original and actual execution times. Let d denote the delay time. When the token moves to the next service facility, the inter-arrival time of the next service facility increases by an average of d. The utilization of the M/M/1 queue is defined by $\frac{\lambda}{\mu}$, where λ and μ refer to the arrival and service rates, respectively (Kleinrock 1975). The equation can also be defined by $\frac{s}{a}$, where s and a refer to the service time and inter-arrival time, respectively. Consider the linear queuing network with two queues, and yield statistics at an instant in time. The equation of utilization (ρ_2) for the second queue is defined by equation (1) since the instant inter-arrival time at the second queue is the sum of the service time at the first queue and the delay time by the time interval (*TI*).

$$\rho_2 = \frac{s}{a+d} \tag{1}$$

Let an *error rate* denote the rate of decrease in utilization by the time interval. The error rate e can be defined by equation (2).

$$e = \frac{\rho_2}{\rho_1} = \frac{a}{a+d}, \text{ where } \rho_1 = \frac{s}{a} \text{ and } \rho_2 = \frac{s}{a+d} \tag{2}$$

To calculate an average d, we have to consider the probability P_0 that the service facility does not contain a token. In the open queuing network, the increased number of tokens due to the time interval causes the probability P_0 to drop, thus d increases exponentially. In the closed queuing network, the probability P_0 is not affected by the time interval since all tokens are delayed, reducing the arrival rate to each service facility. All tokens have to wait until the end of the time interval, thus the d of a long-run time-average is $TI/2$. The decline in utilization is affected by half of the time interval. The inter-arrival time of long-run time-average $\bar{a}$ in equation (2) approaches $\bar{s}$, the service time of a long-run time-average. When we substitute $d = TI/2$ into the equation (2), the error rate e is defined by

$$e = \frac{\bar{s}}{\bar{s} + TI/2} \tag{3}$$

The utilization with the time interval, $\rho(TI)$ is defined by equation (3), where TI_0 refers to TI of 0.

$$\rho(TI) = \frac{\bar{s}}{\bar{s} + TI/2} \times \rho(TI_0) = \frac{1}{1 + \mu TI/2} \times \rho(TI_0) \tag{4}$$

Consequently, we can derive the equation to correct the error in utilization. The original value of the utilization in the toroidal queuing network can be approximated by

$$\rho(TI_0) = (1 + \mu TI/2) \times \rho(TI) \tag{5}$$

Figure 7 shows an error rate comparison between the experimental and calculated results. As the ratio of the MST to TI increases, the difference between the two results decreases.

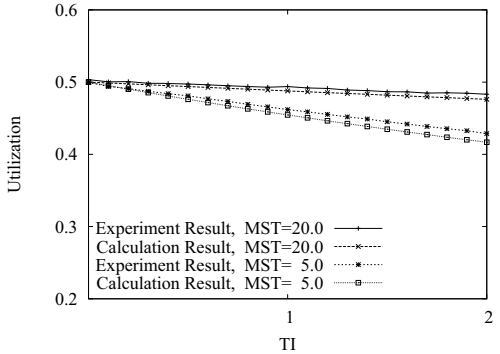


Figure 7: Experimental and calculated results

The equation of the utilization for the error correction is not derived from the analysis of individual nodes. Our intention is to approximate the total error rate when adding one more parameter, time interval so that the error is corrected to yield more accurate results. The equation for the total error rate is easily derived from the existing equations of queuing theory. The equation combined with the results from the simulation produces more accurate results without building a complicated analytical model from each node.

6 CONCLUSION AND FUTURE WORK

We have introduced a new method for simulating queuing models based on a SIMD scheme. There has been little research in the use of a SIMD platform for parallelizing the simulation of queuing models in particular. The concerns in the literature regarding event distribution and the seemingly inappropriate application of GPUs for discrete event simulation are addressed (1) by allowing events to occur at approximate boundaries at the expense of accuracy, and (2) by using a detection and compensation approach to minimize the error. Our hypothesis regarding this research is that the SIMD architecture (as exemplified by the GPU) is useful for *approximate discrete event simulation*.

One of the problems in the cluster experiments was that the communication delay was relatively large compared to a traditional parallel simulation scheme. However, SIMD hardware is designed to minimize the communication cost, and the GPU experiments show that the effect of the communication delay is very small. The GPU experiments show that some parts of codes, the event scheduling, should be parallelized so that we can more comprehensively harness the computational power of the GPU. We are currently implementing the event scheduling method for the GPU. We are currently planning research regarding the implementation of a significant real-world application using our algorithm and analytic approach. We also plan to study optimization approaches for allowing the user to "dial in" a desired accuracy/performance tradeoff.

REFERENCES

Ayani, R., and B. Berkman. 1993. Parallel discrete event simulation on simd computers. *Journal of Parallel and Distributed Computing* 18 (4): 501–508.

Bagrodia, R. L., and M. Takai. 2000. Performance evaluation of conservative algorithms in parallel simulation languages. *IEEE Transactions on Parallel and Distributed Systems* 11 (4): 395–411.

Bolch, G., S. Greiner, H. de Meer, and K. S. Trivedi. 2006. *Queueing networks and markov chains : Modeling and performance evaluation with computer science applications*. New York, NY: Wiley-Interscience.

Fishwick, P. A. 1992. Simpack: getting started with simulation programming in c and c++. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 154–162. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Fujimoto, R. M. 1990. Performance of time warp under synthetic workloads. In *Proceedings of the SCS Multi-*

conference on Distributed Simulation, Volume 22, 23–28. San Diego, CA: Society for Computer Simulation International.

Fujimoto, R. M. 1999. Exploiting temporal uncertainty in parallel and distributed simulations. In *Proceedings of the 13th workshop on Parallel and distributed simulation*, 46–53. Washington, DC: IEEE Computer Society.

Fujimoto, R. M. 2000. *Parallel and distribution simulation systems*. New York, NY: Wiley-Interscience.

Gobron, S., F. Devillard, and B. Heit. 2007. Retina simulation using cellular automata and gpu programming. *Machine Vision and Applications* 18 (6): 331–342.

Harris, M. J., W. V. Baxter, T. Scheuermann, and A. Lastra. 2003. Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 92–101. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

Kiesling, T., and S. Pohl. 2004. Time-parallel simulation with approximative state matching. In *Proceedings of the 18th workshop on Parallel and distributed simulation*, 195–202. Los Alamitos, CA: IEEE Computer Society.

Kleinrock, L. 1975. *Queueing systems volume 1: Theory*. New York, NY: Wiley-Interscience.

Kumar, P., and S. Radha. 2007. Parallel discrete event simulation of ieee 802.11 mac layer using cell processor. In *International Conference on Emerging Trends in High Performance Architecture Algorithms and Computing (HiPAAC 2007)*.

Law, A. M., and W. D. Kelton. 2006. *Simulation modeling & analysis*. 4th ed. New York, NY: McGraw-Hill, Inc.

Martini, P., M. Rümekasten, and J. Tölle. 1997. Tolerant synchronization for distributed simulations of interconnected computer networks. In *Proceedings of the 11th workshop on Parallel and distributed simulation*, 138–141. Washington, DC: IEEE Computer Society.

NVIDIA Corporation 2006. *Technical brief: Nvidia geforce 8800 gpu architecture overview*. NVIDIA Corporation.

NVIDIA Corporation 2007. *Nvidia cuda programming guide*. 1.1 ed. NVIDIA Corporation.

Nyland, L., M. Harris, and J. Prins. 2007. Fast n-body simulation with cuda. In *GPU Gems 3*, ed. H. Nguyen, Chapter 31. Upper Saddle River, NJ: Addison Wesley.

Owens, J. D., D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. 2005. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, 21–51.

Patsis, N. T., C. Chen, and M. E. Larson. 1997. Simd parallel discrete event dynamic system simulation. *IEEE Transactions on Control Systems Technology* 5:30–41.

Perumalla, K. S. 2006. Discrete-event execution alternatives on general purpose graphical processing units (gpgpus). In *PADS '06: Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 74–81. Washington, DC: IEEE Computer Society.

Rao, D. M., N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. 1998. Unsynchronized parallel discrete event simulation. In *Proceedings of the 1998 Winter Simulation Conference*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, 1563–1570. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Shu, W. W., and M. Wu. 1995. Asynchronous problems on simd parallel computers. *IEEE Transactions on Parallel and Distributed Systems* 6 (7): 704–713.

Vakili, P. 1992. Massively parallel and distributed simulation of a class of discrete event systems: a different perspective. *ACM Transactions on Modeling and Computer Simulation* 2 (3): 214–238.

Wang, J. J., and M. Abrams. 1992. Approximate time-parallel simulation of queueing systems with losses. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 700–708. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Xu, Z., and R. Bagrodia. 2007. Gpu-accelerated evaluation platform for high fidelity network modeling. In *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, 131–140. Washington, DC: IEEE Computer Society.

AUTHOR BIOGRAPHIES

HYUNGWOOK PARK is a PhD student of Computer and Information Science and Engineering at the University of Florida. He received an M.S. degree in Computer and Information Science and Engineering from University of Florida in 2003. His research interests are modeling and parallel simulation. His email address for these proceedings is <hwpark@cise.ufl.edu>.

PAUL A. FISHWICK is Professor of Computer and Information Science and Engineering at the University of Florida. Fishwick's research interests are in modeling methodology, aesthetic computing, and the use of virtual world technology for modeling and simulation. He is a Fellow of the Society of Modeling and Simulation International, and recently edited the CRC Handbook on Dynamic System Modeling (2007). He served as General Chair of the 2000 Winter Simulation Conference in Orlando, Florida. His email address for these proceedings is <fishwick@cise.ufl.edu>.