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

AN APPROACH FOR THE EFFECTIVE UTILIZATION OF GP-GPUS IN PARALLEL COMBINED SIMULATION

David W. Bauer Jr.
Matthew McMahon
Ernest H. Page

The MITRE Corporation
7515 Colshire Drive, McLean, VA 22102

ABSTRACT

A major challenge in the field of Modeling & Simulation is providing efficient parallel computation for a variety of algorithms. Algorithms that are described easily and computed efficiently for continuous simulation, may be complex to describe and/or efficiently execute in a discrete event context, and vice-versa. Real-world models often employ multiple algorithms that are optimally defined in one approach or the other. Parallel combined simulation addresses this problem by allowing models to define algorithmic components across multiple paradigms. In this paper, we illustrate the performance of parallel combined simulation, where the continuous component is executed across multiple graphical processing units (GPU) and the discrete event component is executed across multiple central processing units (CPU).

1 INTRODUCTION

Combined simulation is described as a technique for the simulation of a class of systems having properties suitable to both continuous simulation and discrete event simulation, two techniques well known to the simulation community. This combined technique was first proposed by (Fahrland 1970) and the first truly combined simulation software, GASP IV, was made available in 1974 by A. A. Pritsker (Pritsker and Hurst 1973).

Three main approaches are common in combined simulation: support discrete event simulation within the context of continuous simulation (Mitchell and Gauthier 1976, Zeigler, Kim, and Praehofer 2000, Brooks 2005), support continuous simulation from a discrete event simulation context (Pritsker and Hurst 1973, Pegden, Sadowski, and Shannon 1995, Klingener 1996) and truly combined simulation software (Cellier and Blitz 1976). More recent examples using HLA and DEVS include (Borshchev et al. 2002, Nutaro et al. 2007), respectively, and parallel combined simulations have been studied in (Kettenis 1997).

Recently, there has been an increasing interest in the use of graphics processors (GPUs) in simulation (Manocha 2005, Perumalla 2006, Lucas, Wagenbreth, and Davis 2007). An open question is whether you can define an approach that would allow exploitation of GPUs in the context of general discrete event simulation. In this paper, we take a step in that direction, suggesting an approach to exploit GPUs in parallel combined simulation.

Our goal is to study the performance of a parallel discrete event simulator executing on the central processors (CPUs), with support for continuous simulation executing on the graphics processors. Incorporating continuous simulation on the GPU in the context of a discrete event simulation model is accomplished by defining a GPU kernel that executes on the GPU, and invoking that kernel for execution during event processing in Time Warp. The Time Warp optimistic protocol was first proposed by Jefferson and our simulator implementation relies on the technique of reverse computation to support rollback (Jefferson 1985, Carothers, Perumalla, and Fujimoto 1999).

In Section 2.1 we review related modeling efforts performed on GPUs, as well as give a brief review of combined simulation in Section 2.2 . In Section 3 we illustrate how continuous models for the GPU are incorporated into our Time Warp implementation. In Section 4 we indicate our experiment framework, and in Section 5 we study the performance impacts of moving a portion of a purely discrete event model to the continuous simulation paradigm for executing on a GPU. Finally, in Section 6 we indicate our conclusions and propose future work in this area.

2 RELATED WORK

In this section we outline work related to modeling & simulation on graphics processors, as well as prior work in the area of parallel combined simulation.

2.1 GPU MODELING

Current modeling efforts that take advantage of *graphical processing units*, or GPUs, has primarily been focused on model codes that execute solely on the GPU. A multitude of data-parallel model examples can be found in the literature illustrating general concepts such as histograms, FFT and image processing, to more specific models of heat diffusion, parallel Mersienne Twister random number generation, N-bodies and more. In each instance, the parallel performance of the model executing on the GPU illustrated tremendous performance improvements over CPU execution, primarily because the models fit the paradigm of the GPU hardware. GPU devices have a disproportionately small amount of available memory in comparison to CPUs (e.g., 768MB available on the nVidia 8800 GTX versus 32GB on a 4-core, 64-bit compute node), and some research has been conducted on cluster computing using GPU-enhanced compute nodes (Fan, Qiu, Kaufman, and Yoakum-Stover 2004, Müller, Strengert, and Ertl 2006, Müller, Strengert, and Ertl 2007).

The modern GPU architecture is *single instruction, multiple data*, or SIMD, contains multiple stream processors (currently as high as 128) with clock rates in the gigahertz range, and parallel data caches (shared memory). Software support for general purpose programming of GPU devices has evolved and APIs now provide access to threading models, atomic operations and support double-precision (64-bit) floating point operations (although our devices only support single precision). The majority of codes implemented for the GPU contain a high degree of data-parallelism and are computed efficiently.

The SIMD GPU architecture has been optimized to efficiently compute data-parallel codes, and it is still unclear how these devices can be leveraged to support discrete event simulation, though this was studied in (Perumalla 2006). In order to take advantage of the full capabilities of a GPU-enhanced cluster computer, it is our contention that the hardware should be utilized as most appropriate to the algorithms computed. We postulate the most advantageous use of the available CPUs for the discrete portions of the system being modeled, and the available GPUs for the continuous portions.

2.2 Parallel Combined Simulation

Combined simulation as a technique for integrating discrete event and continuous simulation models was introduced in Fahrland's thesis (Fahrland 1970) from 1970, and the first truly combined simulation software, GASP IV, was made available in 1974 by A. A. Pritsker (Pritsker and Hurst 1973).

Recent studies related to parallel combined simulation have illustrated significant performance improvements, such as, Perumalla's μsik discrete event simulator combining the conservative and optimistic simulation protocols (Perumalla 2005), Liljenstam's et. al. mixed abstraction level network models (Liljenstam, Yuan, Premore, and Nicol 2002), and the synchronization mechanism for BGP networks in Syzmanski's Genesis simulator (Szymanski, Liu, and Gupta 2003).

Prior work indicates a straightforward speedup in the execution time of a model ported wholly from CPU to GPU, dependent primarily on the number of available processors on the GPU. For example, a 16-fold improvement was reported for a model of heat diffusion in a 2D plate, porting the model from a single CPU to a GPU with 16 vertex processors (Perumalla 2006).

For combined models, we expect similar performance improvements for the continuous portion of the DES model that can be shifted to the GPU for computation. In accordance with Amdahl's Law, for the overall combined model we *do not* anticipate speedup to that degree, as only a portion of the model runtime has been improved (Amdahl 1967). In this paper we report the overall runtime performance improvement incorporating support for continuous simulation on the GPU.

3 OUR APPROACH

Our approach for combining discrete event and continuous models is to define a model that illustrates aspects of both models. We measure the runtime performance using a synthetic workload application that executes in the context of a general-purpose, Time Warp simulation executive. The synthetic application defines an event handler that contains two sections: one related to modeling discretized system, and the second related to modeling a continuous system.

The benchmark application, *PHOLD*, is a derivative of the HOLD model (Vaucher and Duval 1975), extended for parallel discrete event simulation in (Fujimoto 1990). The PHOLD model generates a synthetic workload over

Algorithm 1 The algorithm illustrates the computational granularity for the PHOLD model. The continuous component of the workload will be shifted to the GPU for computation.

PHOLD Computation Grain

```
// Pseudo-code for synthetic workload

// Continuous component
for each element of an array
    compute N instructions,
    store the result in place

// Discrete component
while workload time not exceeded
    ;
```

a range of parameters that allow performance assessments based on application characteristics, rather than an intuitive understanding of a specific application.

The six model parameters are summarized:

1. number of logical processes
2. event population
3. timestamp increment function
4. movement function
5. computational granularity
6. initial event configuration

For most of the experiments conducted, we configure the application to contain 200,000 logical processes, with 1 initial event per logical process (LP) for a total event population of 200,000 events. The timestamp increment function is an exponential distribution with mean equal to 1.0 (parameters 1–3).

The movement function determines the destination LP for an event. Our function constrains the remote event rate over the network to 10% by first drawing a uniformly random number in the range 0..100. If the number drawn is less than the desired remote event rate, then we draw a second uniformly random number in the range $0..N-1$ where N is the number of logical processes, otherwise, we randomly select an LP within our neighborhood as the destination. In our model, the neighborhood is defined by the LPs mapped to a single CPU.

To measure the performance improvement between CPU–GPU, the computational granularity for the combined model is defined by the two components shown in Algorithm 1. The first component models the discrete event computational granularity and is the usual busy wait loop, in microseconds of wallclock time. The second component models a data parallel algorithm that incorporates memory access times and available floating point hardware for given hardware execution environment.

The model introduces parameters related to varying the workload of the continuous component. First, the size of the data array may be varied. The second model parameters indicates the number of operations to be performed per data element.

The third model parameter relates to the threading model of the GPU. The GPU allows the programmer to define the number of threading blocks, and the number of threads per block to allocate. Using multiple threads per stream processor improves performance by masking the latency of the GPU DRAM controller. The GPU is capable of overlapping thread execution on the stream processors when a call to the GPU DRAM is made by a thread.

4 PERFORMANCE STUDY

In this section we benchmark the performance of the parallel combined simulation using the PHOLD model executing in the context of a Time Warp simulation executive. We investigate the performance of the model using both synchronous and asynchronous GPU API functions across a GPU-enhanced cluster of 20 compute nodes. In addition, we report the effects of the parallel combined simulation on the scalability of the Time Warp executive.

4.1 Computing Testbed and Experiment Setup

The Hive GPU-enhanced cluster at MITRE is a Red Hat Enterprise Linux 9.0 cluster consisting of 20 dual-processor, dual-core machines, for a total of 80 cores. The nodes are inter-connected via a dedicated gigabit ethernet switch. Each node's hardware configuration consists of a dual-processor, dual-core AMD Opteron 2200 and 8GBs of main memory. The AMD Opteron 2000-series is a 64-bit chip and provides up to 24GB/s peak bandwidth per processor using HyperTransport technology. The DDR DRAM memory controller is 128-bits wide and provides up to 6.4GB/s of bandwidth per processor. Our RAM configuration consisted of 4 2GB sticks of 400MHz DDR ECC RAM in 8 banks.

Each compute node in the cluster contains a single nVidia 8800 GTX graphics card positioned over two PCI-Express slots which provides 8x2 lanes for a maximum throughput of 4.0 GB/s (at 250MB/s per lane). The GTX is equipped with 768MB of GDDR3 RAM and has a memory bandwidth of 86.4 GB/s. The GTX also contains 128 physical stream processing elements with a clock rate of 1.35GHz.

The main advantage of the nVidia 8800 GTX is that it is one of the first high performance GPU devices to embrace general purpose programming, provide a unified, massively parallel shader design consisting of 128 stream processors with a clock rate of 1.35 GHz. Each processor is capable of processing operations related to vertices, pixelation, geometry or physics and supports IEEE 754 floating-point precision (nVidia 2006). The performance for the nVidia 8800 GTX is 345 GFLOP/s.

4.2 Optimistic Simulation with Reverse Computation

Based on the successful application of *reverse computation* (RC) concepts in other software domains, Carothers, Perumalla and Fujimoto proposed the application of RC to reduce state saving overheads in PDES. They define an approach based on reverse event codes, and demonstrate performance advantages of this approach over traditional state saving for fine-grained applications. For a more comprehensive review of reverse computation, we point the reader to (Carothers, Perumalla, and Fujimoto 1999), and for a performance study of the technique (Carothers, Bauer, and Pearce 2002).

In reverse computation, *destructive* operations are those operations that result in a loss of data. Floating point operations are considered to be destructive, in that operations on these variables result in a loss of precision, and can lead to an incorrect result.

One solution for reverse computing destructive statements is to reverse the computation as a whole, rather than rely on a straightforward application of reverse computing individual statements. For example, this approach was used to avoid state-saving in L'Ecuyers Combined Linear Congruential random number generator (L'Ecuyer and Andres 1997). At a high level, the sequence of seeds generated is based on the following mathematical recurrence:

$$x_{i,n} = a_i x_{i,n-1} mod\ m_i \tag{1}$$

where $x_{i,n} | 1 \leq i \leq 4$ is n^{th} set of four seed values computed from the $n-1$ set of of four seed values, m_i are prime numbers and a_i are primitive roots of m_i. Using the numerical method for computing large powers (Eynden 2001), the inverse of $a_i\ mod\ m_i$ is defined by:

$$b_i = a_i^{m_i - 2}\ mod\ m_i \tag{2}$$

and the reverse seed series is given by:

$$x_{i,n-1} = b_i x_{i,n} mod\ m_i. \tag{3}$$

On a GPU, this approach is greatly simplified for continuous simulation models and is equivalent to hitting the rewind button for a video stream. The programming model for the GPU is data parallel, and so the same instructions for the forward computation can be utilized to generate the reverse computation. The data must be reversed, and not the computation. For example, for the model of N-bodies in space, the trajectory computation for a given body is the same in the forward and reverse, though the velocity and acceleration parameters are negated. What is different is that the computation must be run from the present simulation time t backwards to time $t-1$, multiple times until the correct state is achieved.

5 PERFORMANCE RESULTS

In this section we benchmark the performance of the combined PHOLD model varying parameters across the discrete event and continuous models. We start by studying the GPU-specific parameters related to the GPU threading model, using a single compute node. Moving to ten compute nodes, we now study the effects varying the number of floating point operations and size of the data section in the continuous model. Then we study the GPU asynchronous communication API and complete the analysis with a look at the performance impact on the Time Warp simulation executive.

5.1 GPU Threading Model

The GPU device threading model is broken down into blocks of threads that execute per GPU multiprocessor. A total of 4096 blocks are possible for the nVidia 8800 GTX, with a maximum of 512 threads per block. The nVidia documentation indicates that multiple threads are required to mask the memory latencies that occur between the GPU DRAM and stream processor L1 caches. The cost of fetching a value from the GPU DRAM is an order of magnitude more expensive than performing a floating point operation. Figure 5.1 illustrates the performance of the combined PHOLD model using a variable number of thread blocks and threads per block.

The per LP continuous data segment sent to the GPU defines the maximum number of usable threads. For this experiment, the data segment was fixed at 128KB. The data

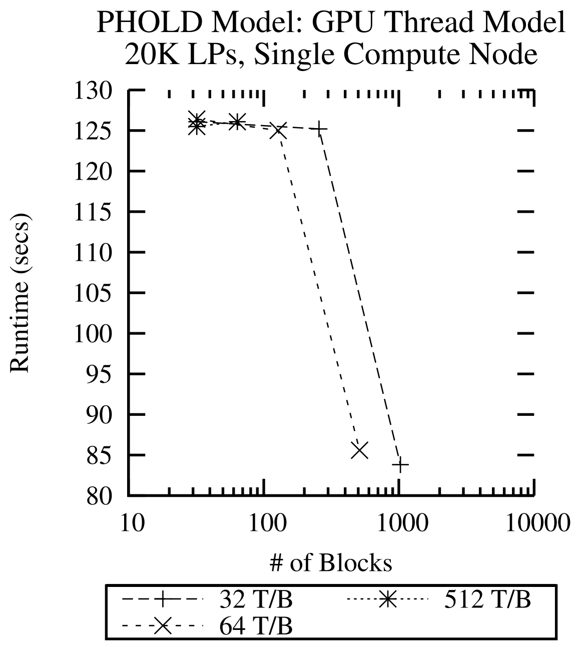

Figure 1: The GPU threading model is allocated as blocks of threads. A maximum of 512 threads per block and 4096 blocks are allowed.

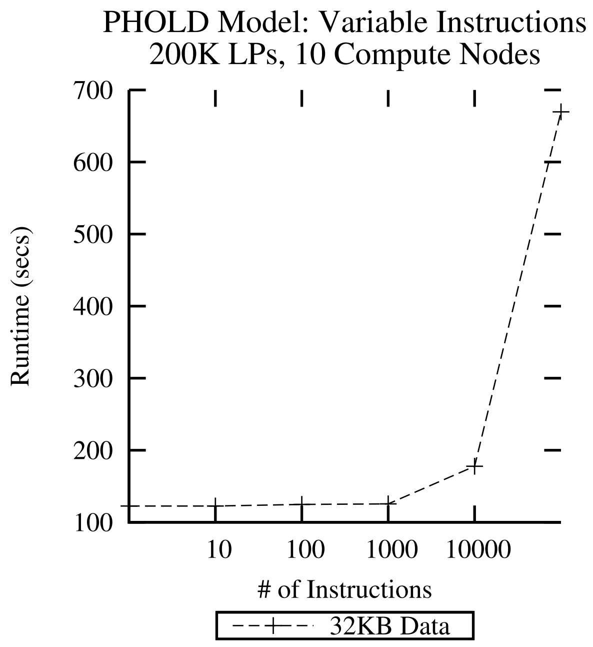

Figure 2: Varying the number of floating point operations performed per data element.

segment is first partitioned by the number of blocks, then by the number of threads per block. Figure 5.1 shows that using more blocks, regardless of the number of threads per block yields the best performance. As indicated by the nVidia documentation, more blocks allows more concurrent execution on the stream processors, and so the 32 T/B (threads per block) case with 512 blocks outperforms the 512 T/B case using only 32 blocks.

5.2 GPU Floating Point Instructions

Next we vary the number of floating point instructions performed per LP data segment variable. We keep the data segment fixed at 128KB, and fix the threading model at 32 T/B for 1024 blocks. We vary the number of instructions per data element to determine the impact of the delay of the continuous model on the overall simulation runtime. For these experiments we utilized 10 compute nodes, and one CPU per node, for a total of 200,000 LPs.

Figure 5.2 shows that unless the number of floating point operations per data variable is very high (e.g., greater than 100,000), then the impact of the GPU performance is minimal. Our hypothesis is that the additional instructions are masked by communication latencies occurring between the CPU–GPU.

Note that it is well accepted that the GPUs 128 stream processors will outperform *any* CPU given a large enough sequence of instructions, so no comparison is performed.

5.3 GPU Memory I/O

The default GPU communication API is fully synchronized and the input-output (I/O) channel between CPUs and GPUs is serialized for all GPU API calls. Calls to the GPU API that block the CPU from performing other work. Communications to the GPU device arrive and are processed in a first in, first out (FIFO) order.

We vary the amount of memory communicated between the CPU LPs and the GPU device. We vary the size of the per LP data segment from 32KB to 128KB. We choose 32 threads per block across 512 blocks for the threading model.

Figure 5.3 indicates that the amount of data communicated between CPU–GPU has a major impact on performance. Intuitively, as the data segment size grows, the communication latencies have less of an impact on the overall runtime.

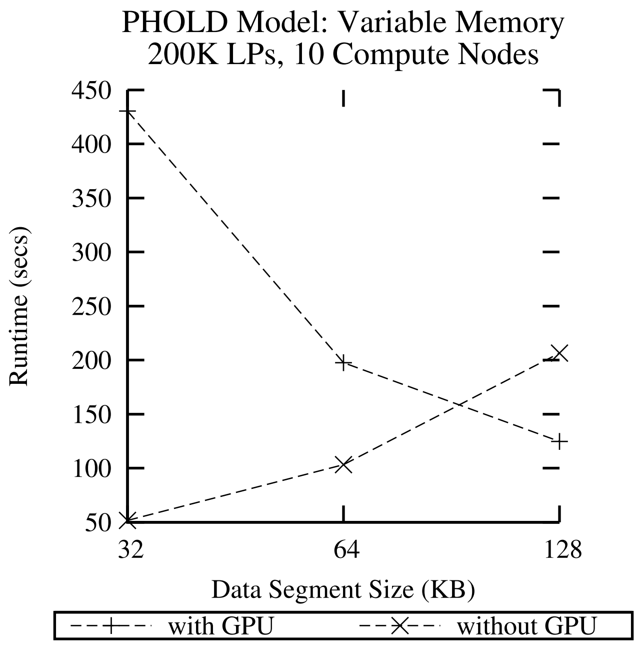

Figure 3: Varying the size of the per LP data segment.

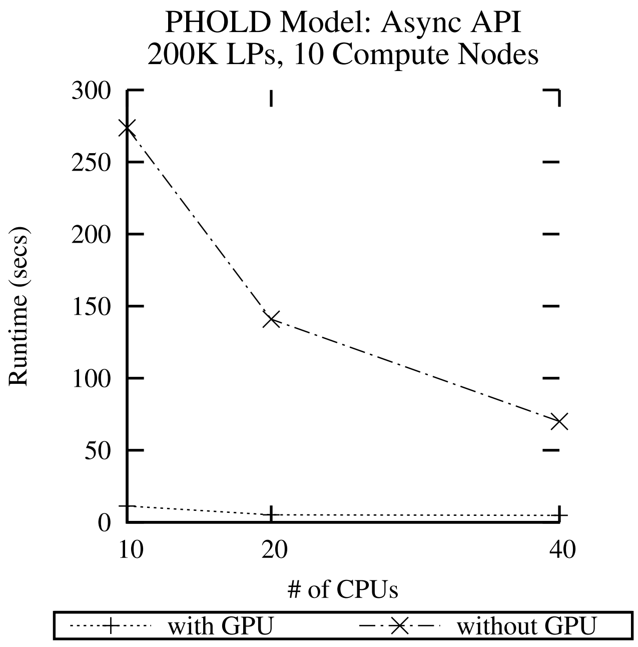

Figure 4: Comparing CPU–GPU runtime performance using the asynchronous GPU API.

5.4 Asynchronous CPU-GPU Utilization

In this section we benchmark the performance of the PHOLD model using the asynchronous GPU communication API. Now as calls are made for memory copies and processing on the GPU, the API immediately returns control to the CPU. This allows for concurrent execution of the combined models, where the computational granularity is defined by the maximum running time between the CPU and GPU codes.

To use the asynchronous API, the nVidia CUDA drivers require any memory communicated to the GPU to be page locked. Turning page locking on improves the performance of the model, but limits the total amount of memory that can be allocated. While this did not cause us to change the size of the model computed, we did not have page locking turned on for the previous results.

For these results, we varied the number of CPUs, using 1, 2 and 4 processors per compute node. Also, we used a per LP data segment size of 128KB, and 100 instructions per array element. Figure 5.4 illustrates significantly improved performance as the I/O channels between the CPU–GPU are more fully utilized by the additional processors calling the device asynchronously.

The speedup results shown in Figure 5.4 indicate that the scalability of the GPUs is limited in comparison to

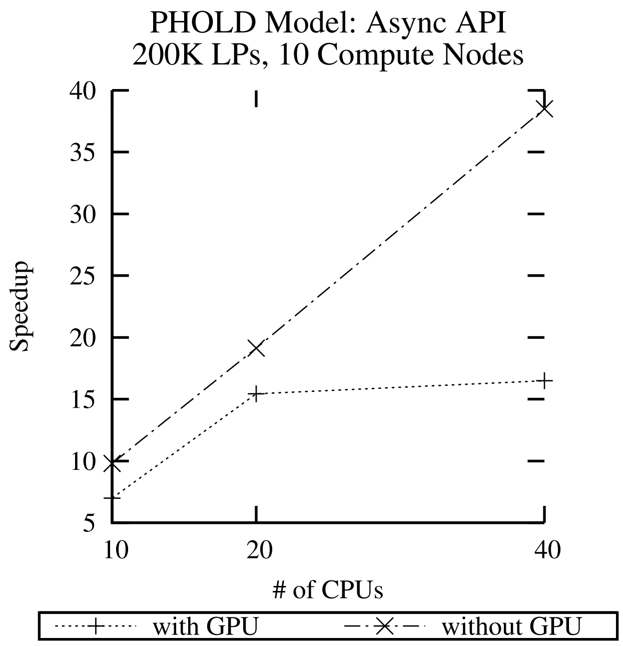

Figure 5: CPU–GPU Speedup Comparison.

CPUs. However, this model has simply become too small for the amount of hardware available and so it is difficult

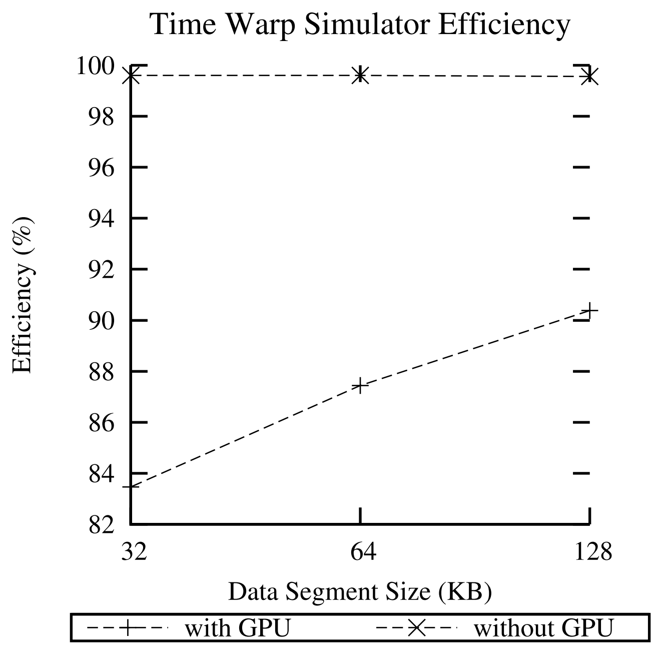

Figure 6: Efficiency of Time Warp simulator with and without GPU support.

for the GPUs to continue to scale. Speedup for the model without CPU was computed using the sequential runtime using a single CPU only, while the GPU-based speedups are based on the runtime for a single CPU–GPU pair. When we compute the speedup for the GPU results based on a single processor and *no* GPU, then we obtain a 539-fold improvement.

5.5 Time Warp Efficiency

In this section we indicate the impact of the combined simulation of continuous and discrete event models on the efficiency of the Time Warp simulation executive. Efficiency is defined as the ratio of events processed in the sequential versus parallel execution of the model. Recall that in the parallel execution of the model, some events may be executed that violate the causality constraint and require rollback to ensure the correct execution of the model.

Figure 5.5 shows that more events are rolled back when the GPUs are invoked. Even though the efficiency of Time Warp with GPUs is 10% less efficient, the overall runtime is reduced when the continuous model is sufficiently large. Runtimes for these experiments are from Section 5.3.

6 CONCLUSIONS & FUTURE WORK

In conclusion we report that the new generation of multi-core, general purpose GPU devices and associated libraries are relatively straightforward to program. However, getting the full performance from the device remains a challenge, specifically, keeping the I/O pipeline full. For parallel combined simulation we believe that GPUs can be utilized effectively for those models with either a large number of continuous variables, or a large amount of floating point operations per continuous variable.

In the future we would like to investigate computing continuous models of physical systems on GPUs, combined with discrete event models executing on the CPUs. We would also like to study additional memory models for the GPU with the goal of improving concurrency between CPUs and GPUs.

REFERENCES

Amdahl, G. 1967. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings* 30 (8): 483–485.

Borshchev, A., Y. Karpov, and V. Kharitonov. 2002. Distributed simulation of hybrid systems with anylogic and hla. *Future Gener. Comput. Syst.* 18 (6): 829–839.

Brooks, C. 2005. *Hyvisual: A Hybrid System Visual Modeler*. Electronics Research Laboratory, College of Engineering, University of California.

Carothers, C. D., D. W. Bauer, and S. O. Pearce. 2002. Ross: A high-performance, low memory, modular time warp system. *Journal of Parallel and Distributed Computing*.

Carothers, C. D., K. S. Perumalla, and R. M. Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Trans. Model. Comput. Simul.* 9 (3): 224–253.

Cellier, F., and A. Blitz. 1976. GASP V: A Universal Simulation Package. *Simulation of Systems: Proceedings of the 8th AICA Congress: Delft, The Netherlands, August*:23–28.

Eynden, C. V. 2001. *Elementary number theory*. McGraw-Hill.

Fahrland, D. 1970. Combined discrete event continuous systems simulation. *SIMULATION* 14 (2): 61.

Fan, Z., F. Qiu, A. Kaufman, and S. Yoakum-Stover. 2004. GPU Cluster for High Performance Computing. *ACM/IEEE Supercomputing Conference 2004*.

Fujimoto, R. M. 1990, January. Performance of time warp under synthetic workloads.

Jefferson, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7 (3): 404–425.

Kettenis, D. L. 1997, February. An algorithm for parallel combined continuous and discrete-event simulation. *Simulation Practice and Theory* 05 (2): 167–184.

Klingener, J. F. 1996. Programming combined discrete-continuous simulation models for performance. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, 833–839. Washington, DC, USA: IEEE Computer Society.

L'Ecuyer, P., and T. H. Andres. 1997. A random number generator based on the combination of four lcgs. *Math. Comput. Simul.* 44 (1): 99–107.

Liljenstam, M., Y. Yuan, B. Premore, and D. Nicol. 2002. A mixed abstraction level simulation model of large-scale Internet worm infestations. *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*:109–116.

Lucas, R., G. Wagenbreth, and D. Davis. 2007. Implementing a GPU-Enhanced Cluster for Large-Scale Simulations. *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*.

Manocha, D. 2005. General-Purpose Computations Using Graphics Processors. *Computer* 38 (8): 85–88.

Mitchell, E., and J. Gauthier. 1976. Advanced Continuous Simulation Language (ACSL). *SIMULATION* 26 (3): 72.

Müller, C., M. Strengert, and T. Ertl. 2006. Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems. Technical report, University of Stuttgart.

Müller, C., M. Strengert, and T. Ertl. 2007. Adaptive load balancing for raycasting of non-uniformly bricked volumes. *Parallel Computing* 33 (6): 406–419.

Nutaro, J., T. Kuruganti, and M. Shankar. 2007. Seamless Simulation of Hybrid Systems with Discrete Event Software Packages. *Proceedings of the 40th Annual Simulation Symposium (ANSS'07)-Volume 00*:81–87.

nVidia 2006, November. Nvidia geforce 8800 gpu architecture overview. Technical Report TB-02787-001_v01.

Pegden, C., R. Sadowski, and R. Shannon. 1995. *Introduction to Simulation Using SIMAN*. McGraw-Hill, Inc. New York, NY, USA.

Perumalla, K. S. 2005. μsik : A micro-kernel for parallel/distributed simulation systems. In *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, 59–68. Washington, DC, USA: IEEE Computer Society.

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, USA: IEEE Computer Society.

Pritsker, A., and N. Hurst. 1973. GASP IV: A combined continuous-discrete FORTRAN-based simulation language. *SIMULATION* 21 (3): 65.

Szymanski, B., Y. Liu, and R. Gupta. 2003. Parallel network simulation under distributed Genesis. *Parallel and Distributed Simulation, 2003.(PADS 2003). Proceedings. Seventeenth Workshop on*:61–68.

Vaucher, J., and P. Duval. 1975. A comparison of simulation event list algorithms. *Communications of the ACM* 18 (4): 223–230.

Zeigler, B., T. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation*. Academic Press.

AUTHOR BIOGRAPHIES

DAVID W BAUER JR is a Lead Modeling & Simulation Systems Engineer at the MITRE Corporation. He earned a Ph.D., M.S., and B.S. from Rensselaer Polytechnic Institute in 2005, 2004, and 2000, respectively. Prior to joining MITRE, he was a research scientist with AT&T and GE. His research interests include parallel and distributed systems, cloud computing, large-scale simulation, wired and wireless networking, and computer architecture. His e-mail address is <bauerd@cs.rpi.edu>.

MATTHEW MCMAHON is a Lead Modeling & Simulation Engineer at The MITRE Corporation. He holds a Bachelor of Electrical Engineering degree from Auburn University, and a Master of Science degree in Computer Science from Virginia Tech. He is a member of IEEE and of the ACM. His email address is <mcmahon@mitre.org>.

ERNEST H PAGE is a member of the technical staff for The MITRE Corporation. He received the Ph.D. in Computer Science from Virginia Tech in 1994. He serves on the editorial boards of SCS Simulation, SCS Journal of Defense Modeling and Simulation, and the Journal of Simulation. He has served as the ACM SIGSIM representative to the WSC Board of Directors since 2001, and currently holds the position of chair. His email address is <epage@mitre.org>.