Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

PERFORMANCE PREDICTION OF LARGE-SCALE PARALLEL DISCRETE EVENT MODELS OF PHYSICAL SYSTEMS

Kalyan S. Perumalla
Richard M. Fujimoto
Prashant J. Thakare
Santosh Pande

College of Computing
Georgia Institute of Technology
801 Atlantic Dr. NW
Atlanta, GA 30332-0280, U.S.A.

Homa Karimabadi
Yuri Omelchenko
Jonathan Driscoll

SciberQuest Inc.
777 South Pacific Coast Highway
Suite 108 (A)
Solana Beach, CA 92075, U.S.A.

ABSTRACT

A virtualization system is presented that is designed to help predict the performance of parallel/distributed discrete event simulations on massively parallel (supercomputing) platforms. It is intended to be useful in experimenting with and understanding the effects of execution parameters, such as different load balancing schemes and mixtures of model fidelity. A case study of the virtualization system is presented in the context of plasma physics simulations, highlighting important virtualization challenges and issues, such as reentrancy and synchronization in the virtual plane, and our corresponding solution approaches. A trace-based prediction methodology is presented, and is evaluated with a 1-D hybrid collisionless shock model simulation, with the predicted performance being validated against one obtained in actual simulation. Predicted performance measurements show excellent agreement with actual performance measurements on parallel platforms containing up to 512 CPUs.

1 INTRODUCTION

Evaluating and optimizing the efficiency of a parallel/distributed discrete event simulation (PDES) programs on large supercomputer configurations is problematic. For example, different partitioning schemes might need to be evaluated over multiple application scenarios and/or different supercomputer configurations. Because supercomputing CPU time is limited and expensive relative to that on desktop computing platforms, it is desirable to avoid utilizing supercomputing time to experiment with different parallelization and optimization strategies (e.g., tweaking the load balancing algorithm). A much more desirable approach is to execute the applications on a "virtual" supercomputer in order to complete the bulk of the experimentation and performance optimization prior to executing the code on the supercomputer. Since the virtual execution of large numbers of processors cannot itself be accommodated on a single host processor because of time and/or memory constraints, the virtual execution itself should be run on a less expensive parallel platform such as a cluster of workstations.

Virtualization offers many benefits, including the following: (1) repeatable execution (2) a dedicated, easily customizable virtual supercomputer configuration, (3) ready access to large supercomputer configurations, and (4) ease of adding instrumentation without perturbing performance metrics.

A virtual, predictive execution is said to experience a slowdown if it runs slower than actual simulation on same number of processors, i.e., the virtual execution takes more total time than the application would have taken to execute on the supercomputing platform. However, it is worth noting that virtual execution-based prediction is still useful even if it experiences such a slowdown. This is because the virtual execution offsets the loss of performance with its several other desirable properties – it retains repeatability of parallel execution and cleanliness of target platform, without the need for actual (supercomputing) platform for execution.

1.1 Motivation

The need for virtualization of PDES applications is exemplified by one specific project (Karimabadi 2005) concerning the simulation of the Earth's global magnetosphere using PDES techniques. The PDES models require execution on large parallel platforms, and hence we are considering supercomputing platforms for this purpose. However, the computational models of the Earth's magnetosphere reflect its strong heterogeneity, and the best parti-

tioning schemes for parallel execution are not known a priori. Thus, we are faced with the need to evaluate different partitioning schemes, as well as other aspects such as accuracy and speed tradeoffs for different model approximations – (a) magneto-hydro dynamics, (b) hybrid, and (c) particle-in-cell models. Various metrics need to be measured in order to obtain a qualitative understanding of the model's execution, as the PDES models themselves are novel and need to be better understood. Performance predictions are required to help tune and fine-tune the simulations for best use of the supercomputing resources. It is also useful early on in the development to have an estimate of performance to help overall debugging and optimization.

To meet the challenges of this immediate application, we address the problem of virtualizing the core of PDES applications in general. First, we develop a framework, called **PDES^2**, for PDES-based virtual execution of any PDES applications. Then, we use the framework and apply it to evaluate the performance and instrumentation of PDES models of Earth's magnetosphere.

1.2 Related Work

Traditional performance prediction efforts focused on platform performance evaluation, such as predicting the overall runtime of an application on a parallel machine. For example, traditional direct execution-based efforts focused on predicting how much faster a set of parallel programs would execute on a future parallel computer configuration (Dickens et al 1996, Reinhardt et al 1993). As such, instrumentation of applications is not the primary focus – the emphasis lies more on the parallel computer configuration rather than on the application.

In contrast, the focus of this present work is on prediction and analysis of the causes of performance effects and gathering detailed information about *application-level factors* contributing to observed parallel performance. For example, it is useful to understand which portions of the grid-based physics models contribute to increased event processing loads on certain processors, which grid cells incur more event processing loads relative to other cells, and how blocking time due to distributed time synchronization affects overall performance. This is more akin to debugging and testing the performance of parallel programs, albeit on *virtual* parallel platforms.

Another distinguishing factor is in the class of parallel programs considered. Traditional performance prediction of generalized parallel programs attempts to accommodate a range of general parallel programs, such as MPI-based applications (Prakash and Bagrodia 1998). In our work, the main focus is on performance prediction of *parallel discrete event simulations*. This restricted focus allows abstracting much of the hardware details of the virtual platform, including network switch operation, operating system effects and low level effects such as interrupts. Instead, simple models can be used in their place. For example, it is sufficient to model communication delays using simple delay distributions for fairly accurate performance estimates.

Finally, there is a speed vs. accuracy tradeoff involved in different approaches. Other PDES performance predictors use software emulation (Zheng et al 2004) focusing on accuracy by having cycle accurate model of target machine and predicting network performance. In contrast, the approach presented here focuses on faster approximations of the same.

The rest of the document is organized as follows. In the next section, fundamental concepts in virtualizing PDES systems are introduced. In section 3, the implementation details of the **PDES^2** virtual system are described. This is followed by section 4 in which the trace-based methodology is described for developing abstract models of event-stepped plasma physics models. A performance validation study is presented in section 5 to validate the system, approach and results against actual simulation runs on supercomputers. Finally, status and future work are outlined in section 6.

2 PDES VIRTUALIZATION CONCEPTS

2.1 Virtual & Real System Relationships

Let Φ be the original system of interest. Let α be the PDES model of Φ. α executes on N_α processors. Let β be a PDES simulation of α. β is executed on N_β processors. The relationships of these real and virtual systems are illustrated in Figure 1 as three layers. Each layer is a model of the layer above it.

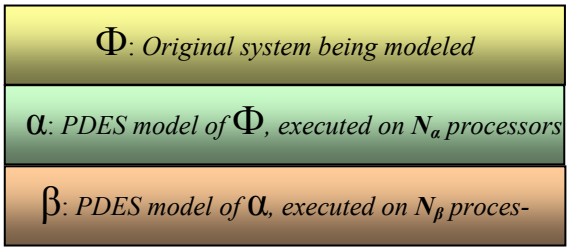

Figure 1: Systems and their Inter-relationships

Note that α and β need not necessarily execute on identical computation platforms. For example, they can be executed on entirely dissimilar types of processors and inter-processor networks. Also, most typically, $N_\alpha \gg N_\beta$. In other words, a large platform execution is virtually realized on a much smaller platform (e.g., execution on supercomputers is simulated on a small cluster of workstations).

2.2 System Timelines

In any PDES execution of a system, there are three fundamental, distinct timelines involved. Additionally, in a PDES of a PDES program, two more timelines appear. Thus, there are five distinct timelines in a PDES of PDES programs, as explained next.

Let $\mathbf{T_\Phi}$ denote a point on **Φ**'s timeline. Let $\mathbf{VT_\alpha}$ be the corresponding virtual time in **α**. Since **α** is a model of **Φ**, $\mathbf{VT_\alpha \equiv T_\Phi}$. Let $\mathbf{RT_\alpha}$ be a point in real-time (wall-clock time) in the execution of **α**. Let $\mathbf{VT_\beta}$ be the corresponding virtual time in **β**. Since **β** is a model of **α**, $\mathbf{VT_\beta \equiv RT_\alpha}$. Let $\mathbf{RT_\beta}$ be the execution time of **β** on same number of processors as **α**. The ratio $\boldsymbol{\rho}=\mathbf{RT_\alpha/RT_\beta}$ at the end of simulation represents the efficiency of virtualization ($\boldsymbol{\rho}<1$ implies **α**'s virtual execution is slower than real execution). These relationships among the various timelines are depicted in Figure 2.

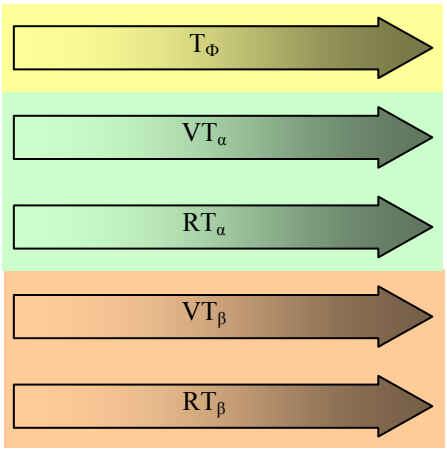

Figure 2: The Five Distinct Timelines Involved in PDES of PDES

For accurate performance prediction, it is important to keep the timelines distinct. Our virtualization framework is carefully designed to keep distinct notions of these times by maintaining separate representations for each at runtime.

2.3 Performance vs. Accuracy

Clearly, there are tradeoffs possible between the performance of the virtual execution and the accuracy of the predicted results. One method is to use analytical models of the application and analyze its execution. Such analytical models are faster to simulate, but are harder to develop as fair approximations of the application. Other methods include virtualizing the entire application context (King et al 2003), which is more challenging than virtualizing abstract models, or developing a virtual model using event traces generated from the original application.

The method chosen for this paper is based on abstract models approximated/tuned with parameters extracted from traces (described in greater detail in section 4).

2.4 Special Synchronization Methods

In plasma simulation, we are investigating the use of special synchronization methods (Karimabadi 2005, Omelchenko 2005) that carefully exploit certain relaxations of accuracy constraints in the models. Our virtual framework is intended to not only explore performance effects of traditional synchronization techniques, but also the relative improvements offered by the novel synchronization techniques.

In particular, we are interested in easily adding and enhancing models of new synchronization methods into the virtual execution. The preemptive event processing (PEP) algorithm (Omelchenko 2005) is of immediate interest as it has been designed to circumvent the low-lookahead constraints of particle-in-cell or hybrid plasma simulations. In later sections, we describe how the PEP synchronization is easily added to virtual execution of general PDES conservative synchronization.

3 IMPLEMENTATION

Our virtualization framework, called **PDES²**, is implemented using a layered approach. The framework is developed over an object-oriented, general-purpose PDES system called μsik (Perumalla 2005) designed for developing efficient PDES applications. The layers in **PDES²** are built using class hierarchy rooted at the threaded simulation process class of μsik.

A *core* layer in **PDES²** is responsible for virtualizing the hardware, modeling the basic PDES synchronization mechanisms of **α**, as well as implementing the PDES synchronization of **β**. This layer is then used to virtualize customized synchronization algorithms of **α**. For plasma simulation, a virtual "preemptive event processing (PEP)" synchronization algorithm is modeled in this layer, as described in detail in later sections. This layer, which we call the virtual PEP layer, is realized as an extension of the core layer.

3.1 Virtual Core Layer

This layer provides the following components:

1. Virtual models of hardware, i.e., abstractions of threaded execution, network latencies, etc., of **α**
2. Models of PDES synchronization, i.e., virtual safe-time computation, etc., of **α**

It also implicitly takes care of synchronization of the PDES execution of **β**. This layer provides a class called virtual CPU (VCPU), which provides a separate stack of execution (maintains a separate thread context) for each VCPU. The following is the class interface of VCPU.

```
class VCPU : public ThreadedSimProcess
{
  virtual void perform_synchronization(VTime vt, VTime dt);

   virtual void begin() = 0;
  virtual void end() = 0;
   virtual void process_next(double *wctime, double *mem) = 0;
   virtual void get_min_ts(VTime *vt, VTime *vdt) = 0;

   int myvcpu_num;
   class LBTSInfo { ... } lbts;
   class ResourceUsage { ... } resources;
}
```

The root VCPU (virtual CPU) class provides all the required support for the virtual context of a single thread of execution. Each instance of VCPU provides a separate thread context (we model only a single thread per CPU, and do not model operating system effects such as context switching).

The virtual CPU thread has a core scheduler loop in its run() method in which it schedules "safe timestamp-ordered" processing. The safe-time at a processor is a lower-bound on the timestamp (LBTS) of incoming events from other processors in the future. The virtual CPU implementation includes built-in support for global asynchronous, distributed, reduction-based safe-time computation. As of this writing, it does not consider transient messages (Mattern 1993), if any are sent, in **α**.

It is important to note that the actual minimum-timestamp values across all virtual CPUs need to be reduced to find their global minimum. The synchronization messages of **α** are translated into events in **β**. Accurate determination of the LBTS values of **α** is necessary, since that determines and uncovers the actual load balance/imbalance in **α**. Note that this is distinct from LBTS values used for parallel simulation execution of **β**.

The distributed reductions follow a logarithmic pattern of communication, which requires dramatically smaller number of synchronization messages to be simulated as compared to an all-to-all pattern, for computing virtual safe-time values in **α**.

3.2 Virtual PEP Layer

In this layer, the synchronization framework of the core layer is extended by adding the special PEP functionality, namely, "pulling down" event timestamps whenever a new LBTS is computed. PEP essentially is a distributed algorithm designed to uncover concurrency of event-based distributed simulation of physics models while conserving flux.

Since traditional lookahead-based techniques fail due to potentially zero-lookahead events possible in such simulations, new techniques such as PEP are needed to uncover concurrency without losing flux conservation. PEP does this by safely reducing (pulling down) the timestamps of local pending events based on global information of minimum time delays of all events. Such pulling down of timestamps serves to enable new processable events that would otherwise be un-processable in a traditional zero lookahead conservative parallel simulation.

```
class PEP_VCPU : public VCPU
{
   virtual void begin();
   virtual void perform_synchronization(VTime vt, VTime dt);
   virtual void pulldown_timestamps(VTime vt, VTime dt);
   virtual void process_next(double *wctime, double *mem);
   virtual void get_min_ts(VTime *vt, VTime *vdt);
   virtual void end();

   EventPQ event_pq;
   virtual void schedule( VEvent *ve );
   virtual void deschedule( VEvent *ve );
}
```

The PEP_VCPU class overloads the perform_synchronization() method of VCPU, and implements PEP's pulling down of timestamps in it. Since VCPU::perform_synchronization() models the computation of LBTS in **α**, the pulling down of timestamps can be naturally performed immediately after the new LBTS is known.

3.3 Virtual PEP-Application Layer

In this layer, specific applications of PEP synchronization are developed. The application is initialized in its begin() method, and finalization is done in its end() method. Events are scheduled by using the PEP_VCPU's schedule() method. Each event has a pointer to a Cell instance, representing that the event is scheduled on behalf of that cell. The PEP_VCPU class automatically "pulls down" the timestamps of these events as/when needed, according to the PEP algorithm. When it is time to execute an event, the PEP_VCPU invokes the execute_vevent() method of that event's Cell. Application-specific extensions to the virtual CPU are realized as subclasses of PEP_VCPU, and extending its begin() and end() methods.

3.4 Abstract Virtual PEP Application

An abstract model of a PEP-based application can be virtualized as follows. The Chombo package (Colella 2005) is used to organize the volumes and other data structures in the grid-based model in terms of discrete-space boxes.

The domain is partitioned into B boxes, and mapped to P processors, as shown in Figure 3.

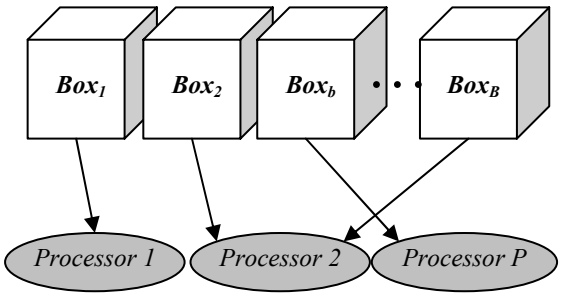

Figure 3: Box to Processor Mapping in Chombo

Each box Box_b is mapped to a processor p_b. The setup is done during initialization in each PEP_VCPU instance. Each box is further organized in terms of grid data elements as shown in Figure 4.

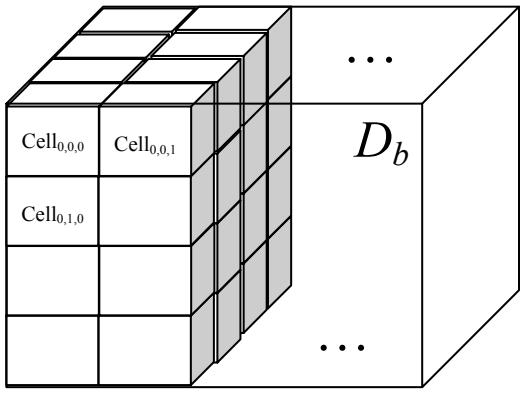

Figure 4: Grid-based Cell Modeling using Chombo

D_b is the LevelData item associated with Box_b. Each D_b is in fact a BaseFab<Cell>. As with any BaseFab, each element within the BaseFab is associated with specific coordinates i,j,k in the domain of the containing box. In our case, each element of the BaseFab is a Cell. Each Cell within a data item D_b, thus, is identified by its coordinates i,j,k within the domain of Box_b. Each Cell contains a priority queue of particles, as depicted in Figure 5.

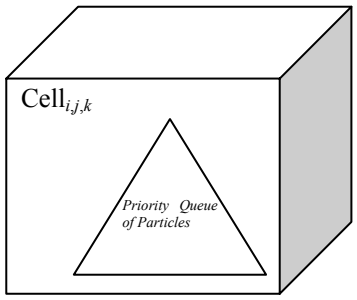

Figure 5: Cell Model Containing Priority Queue of Particles

Each Cell schedules two events into the event queue of the PEP_VCPU instance to which the cell belongs. One event corresponds to the next time of its field update. The other event corresponds to the time of the earliest particle update in the cell. These events' timestamps are subjected to PEP if/as necessary, and later get scheduled by PEP_VCPU. Upon processing one of these events, the Cell re-schedules the same event appropriately into the future.

3.5 Virtualizing Chombo

The Chombo library was originally designed to be used as a "single instance per process," linked to an executable in a single address space. However, virtualized execution requires that multiple instances be realized with the same process address space – one Chombo instance for each virtual CPU. In order to use multiple logically distinct instances of Chombo framework within the same UNIX process, it is necessary to ensure the instances stay distinct do not overlap and interfere with each other. When the code associated with a particular virtual CPU is active, the environment must be arranged such that Chombo functionality for that (and only for that) corresponding virtual CPU is activated.

This is achieved in **PDES²** via a combination of features. First, Chombo is fortunately written by its authors to be reentrant, which we exploit to keep multiple copies of root Chombo objects distinct as long as the identity of the invoking processor is updated at runtime. Next, Chombo provides a sequential mode of compilation which excludes multi-processor, MPI-based support. This mode is used to compile Chombo to avoid undesirable conflict with virtual CPUs, and with μsik's own MPI communication. Finally, a small set of modifications are incorporated into Chombo's query functions that return the processor ID and number of processors in the system. Instead of returning the default MPI-based processor rank and size, the values corresponding to the virtual execution are returned. The changes are only limited to less than a dozen lines of code.

The PEP_VCPU class invokes this added support for switching among multiple logically-distinct instances of Chombo library, by setting and unsetting the Chombo procid variable dynamically, in direct correspondence to multiple virtual CPU instances.

3.6 Hybrid Shock Simulation

As a benchmark for validating our **PDES²** prediction framework, a 1-dimensional hybrid shock simulation program is used (Karimabadi et al 2005). In this hybrid shock code, ions are treated as macro particles whereas electrons are treated as a fluid (electron moments up to and including temperature are retained).

Hybrid codes are ideally suited for physical phenomena that occur on ion time and spatial scales and where a

kinetic description of the electrons is not required. Maxwell's equations are solved by neglecting the displacement current in Ampere's law (Darwin approximation) that eliminates light waves, and by explicitly assuming charge neutrality.

The model problem uses the piston method where incoming plasma moving with flow speed larger than its thermal speed is reflected off the piston located on the most right hand boundary. This leads to the generation of a shock which propagates to the left. The simulation domain is divided into cells, and the ions are uniformly loaded into each cell.

The original 1-D hybrid shock model is written as a μsik (Perumalla 2005) application, and has been ported to run on up to 512 CPUs of the DataStar supercomputer at the San Diego Supercomputing Center. Since the simulation runs on actual large-scale supercomputing platform, it serves as an ideal test case to verify if the **PDES^2** framework accurately predicts the observed performance. In fact, the same supercomputer is also used for virtual execution as well, albeit on much smaller number of processors. In the largest configuration, up to 16 virtual CPUs are hosted per real CPU, giving 512 virtual CPUs on 32 real CPUs for a high-resolution predictive simulation of the 1-D hybrid code, as described later.

4 TRACE-BASED METHODOLOGY

A trace-based methodology is employed to configure and tune the prediction model to the actual simulation, as illustrated in Figure 6. In this, the actual simulation is first executed sequentially on one processor to generate a runtime trace of all events scheduled by the model. Important aspects of each event included in the trace are: the event's timestamp increment, δ; its elapsed wallclock time for execution, ω; and the event's type (particle motion, field update, etc.).

The "replicated scaling" methodology for extending the model to larger problem sizes makes the process of virtualization easier. The δ and ω time statistics are generated from a single processor execution, and they are replicated on larger no. of processors.

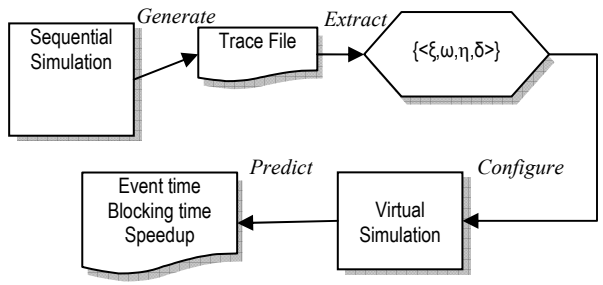

Figure 6: Trace-based Configuration of Virtual Execution

A sequential simulation is used to generate a trace file containing comprehensive event processing statistics, including event types, source and destination cells and computation time per event. From the trace file, a set of tuples is extracted: {<ξ, ω, η, δ>}, where ξ is an event type, ω is wallclock time to process an event of type ξ, η is the number of events generated by each event of type ξ, and δ is the simulation time delay added to newly generated events by an event of type ξ. Note that the total number of event types, ξ, is a constant for any given simulation run. For each event type, ω, η and δ represent random variables. In particular, the trace file provides sufficient information to set the range of low and high values for each variable.

For the hybrid shock application, a uniform distribution between the low and high values seems to fit the distribution well for ω and δ. Also, η always equals unity in the hybrid shock simulation because each cell update event schedules exactly one (next) cell update event, and each field update event schedules exactly one (next) field update event.

5 EXPERIMENTAL STUDY

We now focus on experimental study to validate our virtualization models and trace-driven methodology. As mentioned previously, the 1-D hybrid shock plasma simulation is used as the benchmark in our validation experiments. The aim of the validation experiments is to verify how closely the performance metrics match between the predicted values and actual observed values, for a range of application scenarios.

5.1 Experiment Platform

All performance data reported here are collected on the San Diego Supercomputing Center's IBM DataStar supercomputer. The DataStar is a cluster of 8-way IBM P655 nodes, each node with 8 Power4 1.5GHz processors and 16GB memory (shared by the 8 processors). The nodes are connected by an IBM Federation Switch providing low latency and high bandwidth communication.

For virtual execution, the following number of real CPUs are used for varying number of virtual CPUs (real CPUs→virtual CPUs): 1→1, 2→2, 4→4, 8→8, 8→16, 8→32, 8→64, 16→128, 32→256, 32→512.

For all experiments, the end-to-end communication latency is set to 100 microseconds, based on latency benchmarks run separately. Thus, each message sent across processors in **α** are sent as events in **β** with a simulation time delay of 100 microseconds. Consequently, this delay also serves as the lookahead for conservative parallel execution of **β**.

5.2 Metrics of Interest

Some of the metrics that are of interest for prediction include the total number of events processed by the simulation, and the amount of elapsed/wallclock time consumed by the simulation to reach a certain simulated end-time. The event-counts provide information about overall speed of simulation. The predicted wallclock time gives an estimate of how much supercomputing CPU time is expected to be allocated to simulate a given phenomenon to completion.

In case of ill-balanced scenarios, additional per-processor statistics are useful. Such statistics include per-processor event-counts and time spent blocked in synchronization/communication. In fact, **PDES**^{2} is fully equipped to generate such blocking time statistics, but this feature is not exercised for this document, and detailed experimental numbers for the same are not reported here. However, to facilitate analysis, our virtual framework is indeed capable of producing all such statistics, without perturbing the virtual distributed execution.

5.3 Validating Total No. of Events

Figure 7 shows the prediction of number of simulated events executed within a given simulation end time in **α**. It is seen that the predicted event counts match extremely well with the observed values in actual execution, even across configurations differing in event load by an order of magnitude. The match holds not only along multiple scales of number of processors, but also on the scenario size (number of cells per CPU).

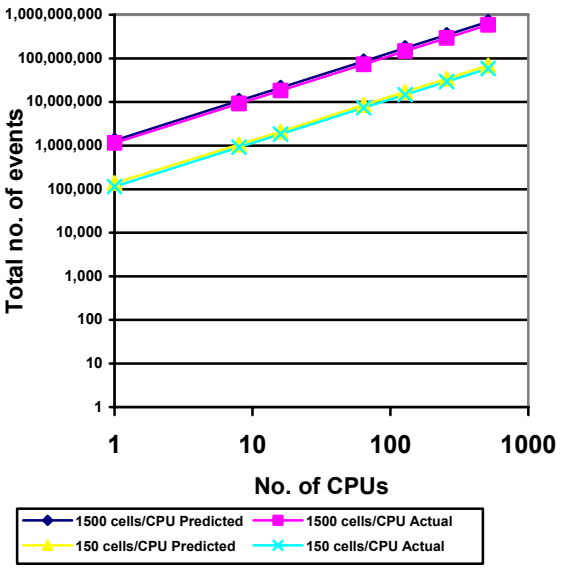

Figure 7: Prediction of Number of Events Executed in α by Simulation End-time of 100.0 in 1-D Hybrid Shock Model

5.4 Validating Wallclock Time

Figure 8 shows the prediction accuracy with respect to wallclock time taken (by **α**). Again, excellent match is observed between predicted and actual performance, except for two values. A runtime aberration in actual execution on 256 CPUs is clearly not captured by the prediction, since no platform artifacts (such as operating system schedulers) are modeled. Also, a slight inversion of relative positions of predicted and actual lines happens on 512 CPUs for 1500 cells/CPU. However, the deviation appears to be well within tolerance levels for the purposes of performance estimation.

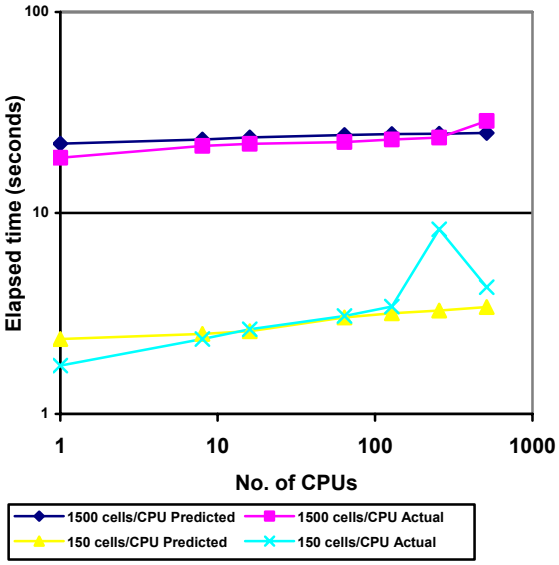

Figure 8: Prediction of Elapsed Wallclock time (RT_α) to Reach Simulation End-time of 100.0 in 1-D Hybrid Shock Model

5.5 Speed of Prediction

Figure 9 shows the speed of prediction relative to the speed of actual simulation, normalized to same number of processors. . Both virtual and actual execution times are normalized to same number of CPUs. Value greater than unity implies virtual execution is faster than actual simulation; less than unity implies virtual execution is slower than actual simulation. Some amount of slowdown is expected since the hybrid shock model is fine-grained.

6 STATUS AND FUTURE WORK

Novel parallel discrete event models in fields such as plasma physics simulations are being developed, aimed at execution on supercomputing platforms. Predictive virtual execution of such models, as opposed to actual simulation, is useful to uncover performance problems ahead of time, without needlessly expending supercomputing resources.

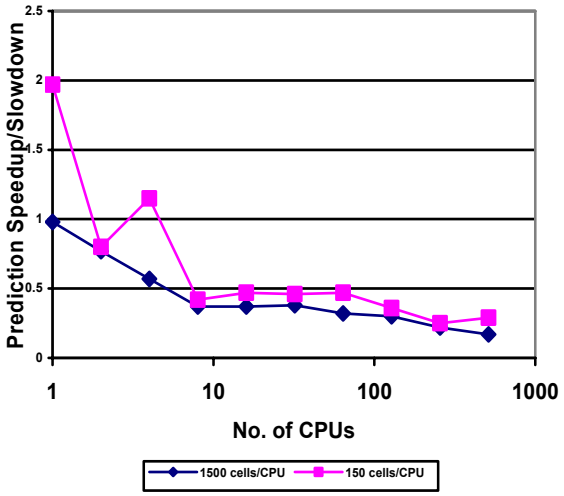

Figure 9: Speed of Prediction on 1-D Hybrid Shock Model Relative to Actual Simulation

Here, a generalized PDES virtualization system, called **PDES²**, for parallel simulation of parallel simulations is presented. The system shows excellent scalability properties, low slowdowns and high predictive accuracy. The system has been validated using a 1-D Hybrid Shock model, which is a complex plasma simulation application. The system is also currently operational on large supercomputing platforms. While retaining high modeling resolution for accuracy, the system also exhibits good scalability, supporting up to 16 virtual CPUs per real/host CPU. The tool is now poised for use in our projects towards investigation of best partitioning strategies and load balancing schemes and their performance effects.

Currently, **PDES²** only supports conservative execution in both **α** and **β** layers. It is desirable to also explore and understand the performance benefits possible with optimistic execution in **α**. Similarly, it is useful to explore speed improvements in the prediction runs by employing optimistic simulation in **β**.

Also, it is currently possible to track memory consumption and hence perform virtual executions to estimate memory requirements. This feature however remains to be tuned and validated against actual simulations.

ACKNOWLEDGMENTS

This work was supported in part at Georgia Tech by the National Science Foundation grant ATM-0326431.

REFERENCES

Colella, P., D. T. Graves, T. J. Ligocki, D. F. Martin, D. Modiano, D. B. Serafini, and B. V. Straalen. 2005. Chombo software package for AMR applications: Design Document. http://seesar.lbl.gov/anag/chombo/.

Dickens, P., P. Heidelberger, and D. M. Nicol. 1996. Parallelized direct execution simulation of message-passing programs. *IEEE Transactions on Parallel and Distributed Systems*, 7: 1090-1105.

Karimabadi, H., J. Driscoll, Y. Omelchenko, and N. Omidi. 2005. A new asynchronous methodology for modeling of physical systems: breaking the curse of courant condition. *Journal of Computational Physics*, 205.

Karimabadi, H., J. Driscoll, Y. Omelchenko, K. S. Perumalla, R. M. Fujimoto, and N. Omidi. 2005. Parallel discrete event simulation of grid-based models: asynchronous electromagnetic hybrid code, *Lecture Notes in Computer Science*, in press.

King, S. T., G. W. Dunlap, and P. M. Chen. 2003. Operating system support for virtual machines, presented at Annual USENIX Technical Conference.

Liu, J., D. M. Nicol, B. J. Premore, and A. Poplawski. 1999. Performance prediction of a parallel simulator, presented at Workshop on Parallel and Distributed Simulation, Atlanta, GA.

Mattern, F. 1993. Efficient algorithms for distributed snapshots and global virtual time approximation, *Journal of Parallel and Distributed Computing*, 18: 423-434.

Omelchenko, Y. 2005. Scientific discrete event simulation (SciDES) tools, SciberQuest Inc., Technical Report.

Perumalla, K. S. 2005. µsik - a micro-kernel for parallel/distributed simulation systems," presented at Workshop on Principles of Advanced and Distributed Simulation, Monterey, CA.

Prakash, S. and R. Bagrodia. 1998. MPI-Sim: using parallel simulation to evaluate MPI programs," presented at Winter Simulation Conference.

Reinhardt, S. K., M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. 1993. The Wisconsin wind tunnel: virtual prototyping of parallel computers," in *Proceedings of the 1993 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 21: 48-60.

Wilmarth, T., G. Zheng, E. J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, and L. V. Kale. 2005. Performance prediction using simulation of large-scale interconnection networks in POSE, presented at Workshop on Principles of Advanced and Distributed Simulation, Monterey, CA.

Zheng, G., G. Kakulapati, and L. V. Kale. 2004. BigSim: a parallel simulator for performance prediction of extremely large parallel machines, IPDPS.

AUTHOR BIOGRAPHIES

KALYAN S. PERUMALLA is a research faculty member in the College of Computing, Georgia Tech. His interests include experimental research in Parallel and Distributed Systems, Network Simulation/Emulation, Security and

Parallel Combinatorial Optimization. He has published over 40 papers on these topics in journals and conferences, and has won three best paper awards. Several of his research prototypes of parallel/distributed simulation systems are disseminated and being used worldwide. Dr. Perumalla received his Ph.D. degree in Computer Science from Georgia Tech in 1999. He can be reached via email at kalyan@cc.gatech.edu, and his homepage is at www.cc.gatech.edu/fac/kalyan.

RICHARD M. FUJIMOTO is a Professor in the College of Computing, Georgia Tech. He received the Ph.D. and M.S. degrees from the University of California (Berkeley) in 1980 and 1983 (Computer Science and Electrical Engineering). He has been an active researcher in the parallel and distributed simulation community since 1985. He has published widely on his research, and has won three best paper awards. Among his current activities, he is the technical lead concerning time management issues for the DoD High Level Architecture (HLA) effort. He is an area editor for ACM Transactions on Modeling and Computer Simulation, and has also been chair of the steering committee for the Workshop on Parallel and Distributed Simulation, (PADS) since 1990, and co-program chair of MASCOTS'05. Dr. Fujimoto can be reached via email at fujimoto@cc.gatech.edu, and his homepage is at www.cc.gatech.edu/~fujimoto.

PRASHANT J. THAKARE is a graduate student in the College of Computing, Georgia Tech. He is a student of Dr. Santosh Pande and is interested in Language and Compiler aspects of simulations. He can be reached via email at thakare@cc.gatech.edu.

SANTOSH PANDE is an Associate Professor in the College of Computing, Georgia Tech. His primary interest is in investigating static and dynamic compiler optimizations on evolving architectures. He has published over 40 papers in journals and conferences. He served as a guest editor and chair of several leading journals and conferences such as PLDI and LCTES. Dr. Pande can be reached via email at santosh@cc.gatech.edu, and his homepage is at www.cc.gatech.edu/~santosh.

HOMA KARIMABADI is the founder of SciberQuest and co-founder of three other startups. He has a unique combination of expertise in supercomputing techniques and information technology as well as strategic analysis and business management. He has published over 50 articles in scientific journals on a wide range of topics including cosmology, chaos theory, space plasmas, numerical algorithms, and solar physics. Dr. Karimabadi is also heading the space physics plasma simulation group at UCSD. He received his B.S. in mathematics and astronomy from UC Berkeley and his PhD in plasma astrophysics from the University of Maryland. He can be reached via email at homa@san.rr.com, and his homepage is www.sciberquest.com/about/?staff_id=3.

YURI OMELCHENKO is a recognized expert in computational plasma physics and high-performance scientific computing. He has made seminal contributions to the theory of ionospheric phenomena, magnetized ion rings and compact field-reversed configurations (FRCs). Dr. Omelchenko developed a number of state-of-the-art, object-oriented codes for modeling space and fusion plasma experiments on massively parallel platforms at Cornell and General Atomics. He spearheaded the development of adaptive mesh refinement (AMR) simulation software for radiation-hardened semiconductor devices at Dynamics Research Corporation. He is leading the design and development of numerical algorithms and distributed infrastructure for novel, self-adaptive, discrete-event simulation (DES) models targeted for a broad range of physics-based applications. Dr. Omelchenko received his M.S. and Ph.D. degrees in applied mathematics and physics from the Moscow Institute of Physics and Technology and completed post-doctoral studies at Cornell University. He can be reached via email at omelche@earthlink.net, and his homepage is at www.sciberquest.com/about?staff_id=4.

JONATHAN DRISCOLL is a software engineer at SciberQuest, Inc.. Prior to his joining ScibrQuest, Inc., he was awarded a fellowship to conduct research on digital signal processing and test with computer simulation. He developed digital signal processing methods for VERITAS ground based gamma ray telescope. At SciberQuest, his primary role has been working on discrete event simulations as well as application of genetic programming techniques to space plasmas. He can reached via email at jdriscoll@copper.net.