ELSEVIER

Simulation Practice and Theory 8 (2001) 529–553

SIMULATION PRACTICE AND THEORY

www.elsevier.com/locate/simpra

A performance and scalability analysis framework for parallel discrete event simulators ☆

Vijay Balakrishnan, Radharamanan Radhakrishnan *, Dhananjai Madhava Rao, Nael Abu-Ghazaleh, Philip A. Wilsey

Department of ECECS, Experimental Computing Laboratory, University of Cincinnati, P.O. Box 210030, Cincinnati, OH 45221-0030, USA

Received 5 June 2000; received in revised form 1 March 2001

Abstract

The development of efficient parallel discrete event simulators is hampered by the large number of interrelated factors affecting performance. This problem is made more difficult by the lack of scalable representative models that can be used to analyze optimizations and isolate bottlenecks. This paper proposes a performance and scalability analysis framework (PSAF) for parallel discrete event simulators. PSAF is built on a platform-independent *Workload Specification Language* (WSL). WSL is a language that represents simulation models using a set of fundamental performance-critical parameters. For each simulator under study, a WSL translator generates synthetic platform-specific simulation models that conform to the performance and scalability characteristics specified by the WSL description. Moreover, sets of portable simulation models that explore the effects of the different parameters, individually or collectively, on the execution performance can easily be constructed using the *Synthetic Workload Generator* (SWG). The SWG automatically generates simulation workloads with different performance properties. In addition, PSAF supports the seamless integration of real simulation models into the workload specification. Thus, a benchmark with both real and synthetically generated models can be built allowing for realistic and thorough exploration of the performance space. The utility of PSAF in determining the boundaries of performance and scalability of simulation environments and models is demonstrated.

Keywords: Parallel discrete event simulation; Performance analysis; Scalability; Synthetic workload generator

☆ Support for this work was provided in part by the Defense Advanced Research Projects Agency under contract numbers DABT63-96-C-0055 and J-FBI-93-116.

* Corresponding author. Tel.: +1-513-556-0904; fax: +1-513-556-7326.
E-mail address: ramanan@ececs.uc.edu (R. Radhakrishnan).

0928-4869/01/$ - see front matter
PII: S0928-4869(01)00033-7

1. Introduction

The complexity of simulation models has rapidly exceeded the capabilities of today's sequential simulation platforms. Fundamental changes to traditional simulation paradigms must be explored to enable the simulation of large models. One of the most promising alternatives is *Parallel Discrete Event Simulation* (PDES) [17], a simulation paradigm that capitalizes on the inherent parallelism present in physical systems. In addition to the potential for improved performance, the resources introduced by a parallel environment increase the capacity of the simulator. Unfortunately, the benefits of parallel simulation have yet to be successfully realized. This failure occurs because of two primary reasons: (i) parallel simulation is highly dynamic and unpredictable and consequently it is difficult to identify and characterize bottlenecks and (ii) there is a shortage of portable, representative models written for parallel simulation environments; it is difficult to evaluate the performance space of new optimizations, and to isolate bottlenecks. Thus, the design, analysis and performance tuning of parallel simulators has been done arbitrarily or based on a restricted set of models (that vary by investigation).

This paper presents a performance and scalability analysis framework (PSAF) for parallel discrete event simulators. PSAF uses a *Workload Specification Language* (WSL) to describe a model in terms of its performance-critical factors. This platform-independent representation can then be translated or ported (using a simulator-specific translator) to different simulation environments. A WSL translator is provided in order to simplify the task of building simulator-specific translators [3]. The translator is constructed using the Purdue Compiler Construction Tool Set (PCCTS) and retargetting (or porting) [1] the translator to different simulators involves the modification of a small set of code generation routines. Thus, synthetic workloads with tunable performance and scalability related parameters are generated. These workloads allow the performance and scalability of a simulator to be investigated under controlled conditions. The analysis is further aided by a *Synthetic Workload Generator* (SWG) which is a program that automatically generates WSL descriptions. The SWG can be used to generate scaled versions of the same WSL description to allow scalability analysis of a particular simulator. The portability of the synthetic models afforded by the system allows unbiased and thorough comparison of simulators.

The motivation for this work stems from the fact that there is a paucity of tools that can be used to compare the scalability and performance of parallel discrete event simulators. Thus, the goal of this work is to provide the simulation community with a standard framework that *simplifies* the analysis of performance and scalability of PDES simulators. Such an analysis allows a simulation system developer to understand the intricacies of the performance related factors that affect the simulation system, for different scales of the model. To that end, a benchmark suite with real

[1] When we say "retargetting" or "porting" we mean that the same simulation model can be executed on different simulators provided a translator exists for each simulator.

(explicitly ported) models, along with synthetic models generated by the SWG, serves to provide the standard framework necessary for the analysis of the performance and scalability aspects of PDES simulators. The aim of this paper is not to present a real benchmark suite for PDES simulators but instead to present a framework that allows users to easily develop a real benchmark suite for PDES simulators.

The remainder of this paper is organized as follows. Section 2 reviews PDES protocols. Section 3 discusses some related efforts in the performance analysis of PDES simulators. Section 4 presents the performance and scalability analysis framework and discusses its various components. Section 5 presents some experiments conducted using PSAF to characterize our own parallel simulator; these experiments provide examples of the different ways that the framework can be used. Section 6 describes the effort needed to retarget the framework to other simulators and presents some comparisons of some PDES simulators. Section 7 investigates the success of the synthetic models in replicating the behavior of their real-model counterparts. Finally, Section 8 contains some concluding remarks.

2. Background

In this section, a brief overview of PDES [17] is presented. In PDES, the model to be simulated is decomposed into *physical processes* that are modeled as *simulation objects*. Each simulation object is assigned to a *Logical Process* (LP); the simulator is composed of a set of LPs concurrently executing their simulation objects. Simulation objects communicate by exchanging time-stamped messages through the LPs. Thus, each LP (which can be associated with multiple simulation objects) receives messages from other LPs and forwards them to the destination objects. In order to maintain causality, LPs must process messages in strictly non-decreasing time-stamp order [23,28]. There are two basic synchronization protocols used to ensure that this condition is not violated: (i) *conservative* and (ii) *optimistic*. Conservative protocols [7,10,31] strictly avoid causality errors, while optimistic protocols, such as Time Warp [17,23] allow causality errors to occur, but implement some recovery mechanism.

Conservative protocols were the first distributed simulation mechanisms. The basic problem conservative mechanisms must address is the determination of "safe" events. In order to execute an event, the conservative process must first determine that it is impossible for it to receive another event with a time-stamp lower that the event it is currently trying to execute. If the process can independently guarantee this, then the event is deemed to be safe and can be executed without the fear of a causality violation. Processes containing no safe events must block; this can lead to deadlock situations if no appropriate precautions are taken. Several studies on conservative mechanisms and optimizations to conservative protocols have been presented in the literature [7,9,10,29,31].

In a Time Warp simulator, each LP operates as a distinct discrete event simulator, maintaining input and output event lists, a state queue, and a local simulation time (called *Local Virtual Time* or LVT). The state and output queue are present to

support rollback processing. As each LP simulates asynchronously, it is possible for an LP to receive an event from the past (some LPs will be processing faster than others and hence will have local virtual times greater than others) – violating the causality constraints of the events in the simulation. Such messages are called *straggler* messages. On receipt of a straggler message, the LP must rollback to undo some work that has been done. Rollback involves two steps: (i) restoring the state to a time preceding the time-stamp of the straggler and (ii) canceling any output event messages that were erroneously sent (by sending *anti-messages*). After rollback, the events are re-executed in the proper order.

3. Related work

Performance metrics such as speedup [1], scaled speedup [24], and size-up [43] have been proposed for quantifying the performance and scalability of parallel systems. While these metrics are useful for tracking performance trends, they do not provide sufficient information to understand why an application scales poorly on a simulation system. The utility of performance metrics is further limited because they compare only the gross-level performance of the systems. They are not indicative of the details of the performance-critical factors that lead to the speedup figure. Moreover, since speedup is defined with respect to a sequential simulator, it is difficult to obtain speedup figures that are directly comparable to others. Thus, in order to enable better implementations of the simulator, a better characterization of the interrelated performance-critical factors must be made available to developers.

There are several empirical and analytical studies [5,16,18,27,40] of the performance of PDES simulators. Most of these studies are restricted to evaluating the impact of a particular simulator optimization on the performance. The optimization is integrated into an in-house PDES simulator, and the performance of the simulator with and without the optimization using some set of models is reported. The impact of the new optimization is not broadly assessed by such a narrow comparison. In particular, the simulation model used for the comparison significantly influences the performance and scalability. The reported change in performance is also specific to the implementation of memory allocation, event-list management, GVT calculation, and deadlock detection. Accordingly, a set of benchmarks that is representative of the general class of PDES application domain [19] is needed to sufficiently characterize the performance of a simulator under different working conditions.

Ideally, a benchmark suite should contain representative applications from the entire domain of PDES applications. In addition, the models should be deterministic, have a stable behavior, and be easily scalable. The simulation model suggested by Fujimoto [19] consists of a synthetic simulation with a fully connected graph type topology. The model includes the following workload parameters: (i) number of LPs, (ii) message population, (iii) time-stamp increment function, (iv) movement function, (v) computation grain and (vi) initial configuration. By varying each of the parameters, the model can be used to conduct performance analysis of a new simulator configuration. Unfortunately, only a small number of the models described by

Fujimoto [19] are freely available. Moreover, some of the available models have differing implementations.

Several researchers have explored building general performance analysis frameworks for PDES simulators. Gilmer [20] defines some performance parameters of models. The parameters suggested by Gilmer are restricted to communication behavior parameters, and do not capture other important aspects of the models. Reynolds and Dickens [39] SPECTRUM testbed allows a user to implement a simulator configuration (protocol), supply an application, and specify some characteristics of the simulation for observation. The supported observable characteristics include determinism, queuing, processing delays, causality, state change characteristics, balance, activity, and connectivity. The major difference between the SPECTRUM testbed and the framework presented in this paper is that while SPECTRUM defines performance parameters for observation purposes, we generate models from the performance-parameter set in order to explore their effect on performance.

Ferscha and Johnson [15] develop a tool for performance prediction of Time Warp protocols and related optimizations. A Time Warp model is built incrementally and decisions regarding different optimizations are made early in the development stage. Other, similar, test-beds that are currently in use include Yaddes [35] and Maisie [2]. These approaches do not allow a model to be evaluated on different implementations of a simulation protocol. The framework described in this paper provides a mechanism for generating a large number of synthetic applications to test a simulation implementation. Moreover, the representation method can be easily translated to other simulators. The framework allows the user to test and evaluate a large collection of performance and scalability related factors.

Synthetic workload generation has been used for characterizing the performance space of complex systems [26,32]. For example, Kiskis [26] develops a framework (similar to the one suggested in this paper) for the analysis of distributed real-time computing systems. He empirically illustrates that the generated synthetic models are representative of the actual model. Kiskis uses a synthetic workload specification language to describe the structure and behavior of a robotics application and a dual-robot simulator and uses the description to generate synthetic workloads.

4. The performance and scalability analysis framework (PSAF)

This section presents a framework for comprehensive analysis of performance and scalability of PDES simulators. An overview of the framework is shown in Fig. 1. Central to the framework is the WSL, a language for capturing the performance-critical attributes of applications. The first step in using the framework is to specify the workload. A WSL description consists of both real and synthetic workload descriptions. Since the performance-critical attributes are explicitly visible, WSL specifications are used by a (SWG) to generate synthetic workloads. Synthetic workload generation allows the performance space of the simulator to be explored methodically. An automatic translator program translates WSL descriptions to simulator-specific synthetic models. The translator ensures that the synthetic models conform

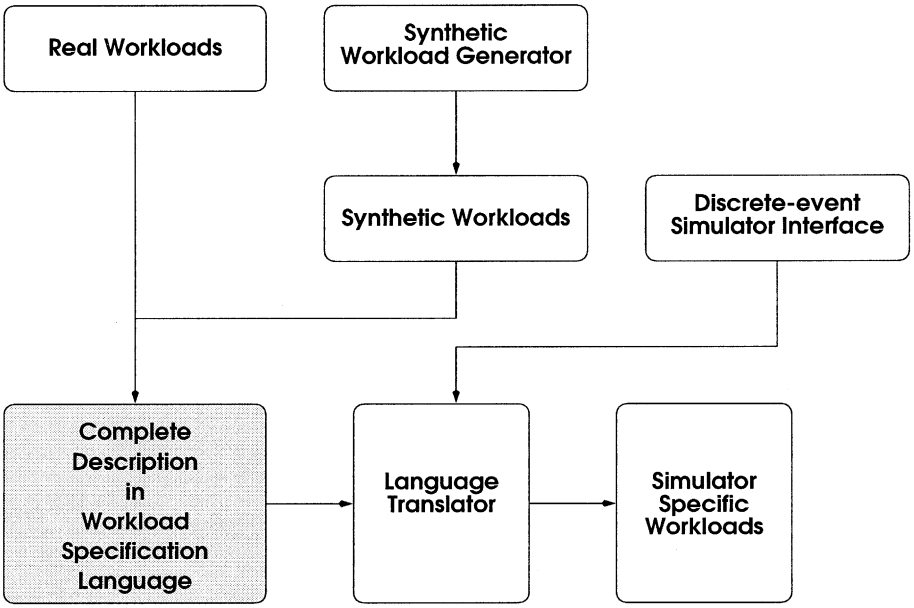

Fig. 1. Performance and scalability analysis framework (PSAF).

to the performance characteristics specified by the WSL description. Real models must be ported manually.

4.1. Workload Specification Language (WSL)

This section describes WSL. WSL is not a modeling language; rather, it is a language for representing the workloads in a format that facilitates performance analysis of parallel discrete event simulators. WSL supports the description of synthetic as well as real workloads. Synthetic workload descriptions are based on a characterization of a PDES workloads in terms of some fundamental parameters. Specification of a real workload is implemented by inserting simulator-specific descriptions (source code) in the WSL object definition. The representation of real models exposes the structure of the models to the user and allows systematic conversion to other simulation languages. While this approach does not reduce the task of the initial modeling of a system, it facilitates translation of models between simulation languages (environments) with high accuracy. In addition, synthetic workload descriptions, written using WSL, can be directly applied to the different simulators (once a WSL translator is written for them).

A workload specification in WSL consists of simulation object definitions followed by instantiations of these objects. Each simulation object may have an unlimited number of definitions. These definitions correspond to the different simulation kernels (real models) or to a synthetic representation. The translator

generates simulator-specific code depending on the target simulation platform. In the remainder of this section, we overview the representation and translation facets of WSL [3,4].

4.1.1. Synthetic representation

A simulation object can be represented as a synthetic object – a representation that does not perform any useful function but produces resource demands that are similar to a real simulation object. Synthetic objects are used to build synthetic workloads. Ideally, the characteristics of the synthetic workloads would match those of real workloads. For this reason, a study of the performance factors that affect PDES simulators was carried out. Fig. 2 illustrates the performance factors of PDES simulators. The performance factors can be classified into six classes. The performance factors in each of these classes must be optimized if maximum throughput is desired. Unfortunately, the classes illustrated in Fig. 2 are not independent of each other. The overall performance is a complex function of all the factors which influence each class. The process of performance analysis can be simplified greatly if each class can be examined separately. From this study, factors that influence the workload were isolated. Several of the parameters have been identified by Fujimoto [17] and

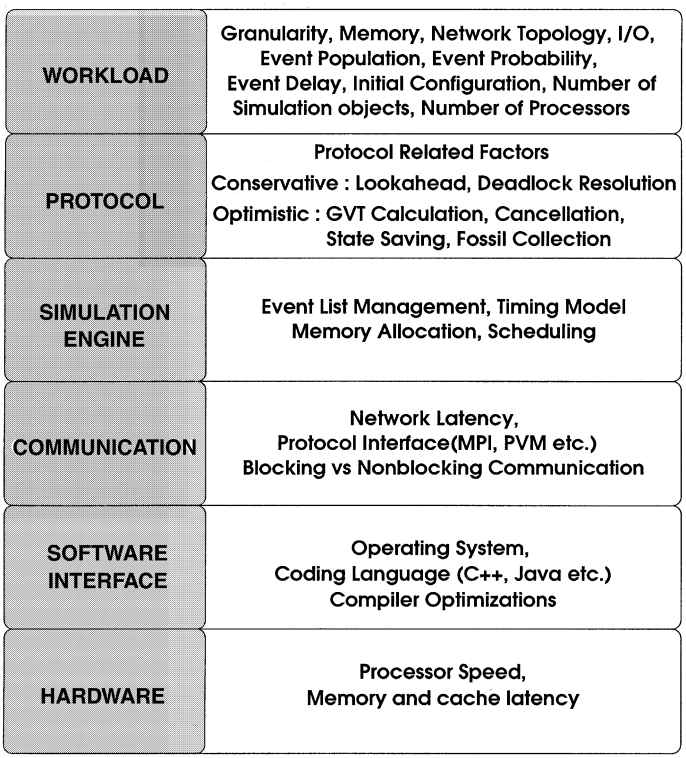

Fig. 2. Performance factors for PDES.

Reynolds [39]. Each parameter is either explicitly defined, or subdivided into more basic attributes to avoid the interdependence among the parameters. The parameters supported by WSL are shown in Fig. 3.

This set of parameters was assembled after studying popular simulation problems used in parallel simulation studies [3]. More precisely, the Shark's World model [12], the Colliding Pucks model [22], the ISCAS'89 digital simulation suite [8], and the Ping Models [5] were ported to our simulation environment and used in determining the set of parameters. The set of parameters is difficult to identify and as we continue to learn more about the behavior of the models, the set of parameters may be changed, and the definitions of the parameters refined. Further analysis of the applications in the PDES domain may reveal other performance-critical parameters. The framework can be easily extended to incorporate additional parameters. Nevertheless, we will demonstrate later in this paper that this set is sufficient to capture the characteristics of the considered models with good accuracy.

The performance parameters identified for PDES workloads form the basis for the representation of synthetic workloads. The syntax of a synthetic description is shown in Fig. 4. The figure also demonstrates the specification of the parameters. A complete model consists of a set of synthetic objects with an associated connectivity pattern (topology). A description of the attributes and their semantics in WSL is given below:

Input: A process either has external input or does not. Objects that have no inputs need to have a special initialization function during simulation. Numbering of inputs is not necessary because, if it becomes necessary to distinguish between the inputs, then we assume that this information will be available in the event. If an object has no inputs then this implies that it cannot receive any events from anyone except itself. To model this, *Input* is a Boolean variable in the WSL description.

Output: This attribute represents the number of output channels associated with this simulation object. It influences other synthetic attributes and is used to determine the number of entries in the lists used to maintain channel attributes. Each

1. Simulation Object related parameters

   [a] Input and Output Behavior [b] Distribution
   [c] Delay [d] State Size

2. Event related parameters

   [a] Computation Granularity [b] Event Size
   [c] Event Population [d] Event Probability

3. Communication related parameters

   [a] Type of connectivity (Topology) [b] Number of Processors[a]
   [c] Number of Simulation Objects

[a]In this paper, the number of processors is always equal to the number of LPs.

Fig. 3. The set of parameters used in PSAF.

```
SimObject nameOfObject {
      SimModel Synthetic {
                  Input boolean ;
                  Output number ;
                  Distribution {distribution}
                               {separation interval list};
                  Delay {distribution, {...} }
                  // One Distribution is needed for each
                  // output channel
            Init   boolean;
            IOFunction   {distribution}
                         {separation interval list};
            Granularity  floating point number;
            EventSize  number ;
            StateSize  number;
            FileInput  number;
            FileOutput  number;
            IterationCount  number;
        }
  }
where
     distribution :=   {UNIFORM | POISSON |
                        EXPONENTIAL | NORMAL| BINOMIAL |  CONSTANT}
                        { seed, seed  }
     separation interval list := a set of pairs of floating point numbers
```

Fig. 4. Structure of a synthetic object.

output channel is assigned a unique number and has the attribute *Delay* associated with it.

Distribution: This is an important attribute that combines several basic parameters. It is an algorithm that describes the production of events for the different output channels. The attribute is specified by selecting a distribution Type. This is generally a random number generator with an associated probability distribution. In addition, intervals are associated with each output channel using a separation interval list. The separation list has an element for each output channel. Every element consists of a list of intervals. For example, if {{a,b} {c,d}}is the separation interval list for an output channel, an event is generated on the output channel is the random number generator produces a number in either of the intervals a to b or c to d.

Delay: This attribute models the delay for each output channel. A distribution generator or constant value is assigned to each output channel. It should be noted that this delay value is not the delay associated with the communication medium. It is the delay used to model the progress of time in the simulation.

Init: For synthetic model descriptions, the initial values are not critical. However, this attribute is supported in order to have repeatable outputs and define correct behavior of the synthetic model. The initial values for synthetic objects is Boolean.

IOFunction: This function defines the probability with which the output changes. This function implicitly takes the current state and the input event. The synthetic model does not necessarily need this function. It is used as a tool for determining the correct behavior of the simulation. It can be used to test whether a sequential simulation of the model produces the same output as the parallel version. Currently supported function are the distribution generators associated with an interval. If the number generated by the statistical function lies in the specified interval then the Boolean output value is inverted. If CONSTANT is defined then the output value is inverted after the defined number of events are processed.

Granularity: This attribute defines the computational granularity of an event. This is a critical attribute; it is important that the meaning of this attribute is kept uniform for all target translators and independent of architectures. The defined granularity does not include the time to schedule the next event nor the granularity introduced by the simulation environment. Granularity is defined in milliseconds and should be translated to the appropriate number of cycles for an architecture. The conversion of this number to the actual number of cycles can be done with the assumption that the same architecture was used for the translation and simulation. Currently, the translator introduces the specified granularity in the event in the form of a loop.

EventSize: The size (in number of bytes) of the events is specified using this attribute. The number specified is the incremental size only. This means that only the size of the event information is counted, but not the additional space introduced by different implementations. This parameter along with the state size and the number of events, models the total memory requirements of the simulation.

StateSize: This attribute represents the size (in number of bytes) of the state of a simulation object. Like *EventSize*, only the state size of the simulation object is recorded. The additional size introduced by specific implementations is not modeled. This parameter can be used to check if the garbage collection routines are performing their operations correctly and in the determination of the rate of garbage collection for a simulation. Rate of garbage collection can then be used for balancing the tradeoffs between speed and memory requirements.

FileInput: This parameter is used to define the number of bytes read by a simulation object every time it is gets an event. This is set to zero if the object performs no reads. This affects the size of the file queues in optimistic protocols because a rollback can create the need for the input to be read again.

FileOutput: This is equal to the number of bytes written out, each time the simulation object is scheduled for execution. This number can be zero for no writes. This attribute affects the size of the file queues in optimistic protocols where only after the event has been committed can the write be made to the file. The *FileInput* and *FileOutput* attributes are used to model the I/O behavior of the simulation object.

IterationCount: This attribute defines the number of events that are to be processed by the simulation object. If this parameter is zero, then the object will not generate events during simulation. This means that if the simulation is to come to a natural stop (as opposed to forcing a termination after a particular time), then all sources must be provided with this parameter and every cycle in the graph (directed graph representing connectivity) must have a source.

4.1.2. Real-model representation

WSL supports real model descriptions in the form of simulation-specific models. Each simulation environment is assigned a unique tag which is used to mark model descriptions specific for it (using the keyword SimModel). When generating real simulation models for a given simulation environment, the translator searches for the tag corresponding to the environment.

The structure of a real object description is shown in Fig. 5. As shown in the figure, the actual descriptions of the target simulation environment are inserted as native code between the double square brackets, *i.e.,* [[...]]. The Event keyword is used to describe the structure of the event and provide initial values to variables in the event structure. The Event description contains the source code for the events that are exchanged between simulation objects. Currently, the translator supports only one event description per simulation model. The State keyword is followed by the source code for the state structure of the simulation object and its corresponding initialization function. The Object description is split into six optional sections. They are:

(a) Variables: Variables or data structures needed to model the simulation object are added here.

(b) Constructor: The code inserted here is translated into a function. If the inserted code was in C++, then this function is equivalent to the constructor for the simulation object class.

(c) Initialize: The code inserted here is executed once before the start of simulation.

```
SimObject nameOfObject {
   SimModel PDESSimulator {
    Input boolean ;
    Output number ;

    Event [[ // declare the variables in the event here ]]
         [[ // initialize the Event variables declared ]];

    State [[ // declare variables needed in the state here ]]
         [[ // initialize State the variables declared here]];

    Object {baseObjectName} {
      Variables [[    // declare any special variables here ]];
      Constructor [[  // constructor (C++ terminology) for the object ]];
      Initialize [[   // put the initialization code here ]];
      Execute   [[    // code for processing the received event goes here ]];
      Finalize  [[    // code to be executed at the end of simulation]];

      [[ // additional functions can be written here ]];
    }
  }
```

Fig. 5. Structure of a real object.

(d) Execute: Code inserted here is executed every time the object is scheduled for execution.

(e) Finalize: Code placed here is executed after the simulation has been completed.

(f) The last section is used for inserting functions that may be called by the code in any of the above sections.

Once the simulation objects have been defined, a *net-list* representing a set of objects and their connectivity is instantiated. The net-list definition enables optional statically-defined simulation object to processor assignments. This feature also allows the study of partitioning algorithms as the translator generates appropriate code to assign simulation objects to different processors. Each object definition may be instantiated multiple times, and connected to other objects using the net-list. Each simulation object (SimObject) may have only one synthetic description but multiple real descriptions. The translator can be directed to choose either synthetic or simulator-specific descriptions.

4.2. Synthetic Workload Generator (SWG)

SWG is a program that automatically generates workloads with emphasis on different performance-related properties. Thus, a suite of models where one or more parameters are varied while the others are held at fixed values can be generated. This enables methodical analysis of the behavior of the simulator with respect to the parameter being varied. Consequently regions of good and bad performance can be easily identified.

SWG operates in two phases: (i) the graph generation phase, and (ii) the model generation phase. The graph generation phase builds a directed graph according to the parameters specified. The controllable graph parameters are the number of nodes and graph topology (e.g., GRID, TREE, COMPLETE, RANDOM). The graph generation phase was built using the Library of Efficient Data-types and Algorithms (LEDA) [30]. Several graph/network topologies can be generated by using the LEDA framework. The generated graph is checked for compliance to additional properties such as number of sources, number of sinks, and the existence of cycles. The second phase starts with the graph representation and converts each node to a synthetic simulation object by filling in the values for the parameters shown in Fig. 4. Once the second phase terminates, the simulation object description and the net-list for the WSL description of the workload are ready.

Fig. 6 shows the structure of the SWG. The parameter values are generated using statistical distributions that are bound at the time of generation. It is possible to override the statistical distribution specification of the parameters, by supplying constant values instead. The language can be extended to model additional distributions.

4.3. WSL translator

The translator is the critical component of this performance analysis framework. The translator translates the synthetic specification into simulation models that mimic the specified behavior on the given simulation system. The translator consists of

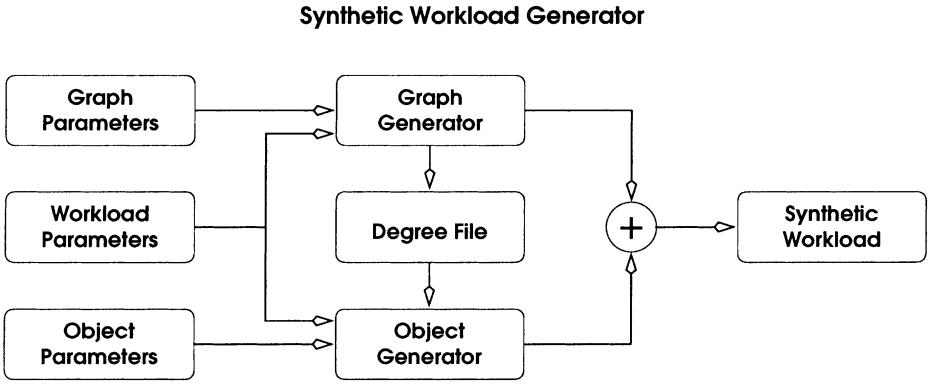

Fig. 6. Structure of the SWG.

a parser that calls predefined functions for each simulation environment supported by the translator. Building a translator for a new simulation environment involves filling in the predefined methods; only the translation phase of the generator needs to be modified. This is discussed in more detail in the following subsection.

Accurate translation of the synthetic objects is crucial to the success of the synthetic models in exercising the system according to the parameter values specified in the WSL description. Each parameter in the synthetic description is well defined, and models that correctly exhibit the required behavior can be built. The translator supports partitioning the network and assigning simulation objects to different processors as required by the WSL specification.

The WSL translator is implemented using the PCCTS [33] framework. A grammar was defined and the PCCTS framework generates a parser that parses an input WSL description [3]. The parser first performs a syntax check on the WSL description before performing semantic checks. During the semantic check, the parser calls "publish" routines from a set of predefined publishing routines. These publish routines write out simulator-specific code. Currently, in PSAF, the publish routines are encapsulated into a *PublishSimulatorCode* class. The publish routines are all defined as member functions of this class. To retarget a translator to generate code for a different simulator, all that is required is the modification of these publish routines.

5. Analysis – using PSAF

In this section, we illustrate the use of PSAF in analyzing some performance and scalability aspects of the WARPED [36] simulation kernel. The models used to perform the experiments were generated by the SWG. The models were then translated to WARPED code using a translator written for WARPED. [2] All the experiments

[2] The WARPED kernel is freely available at http://www.ececs.uc.edu/~paw/warped/

reported in this section were conducted on a SUN SparcCenter 1000 with four processors. All experimental results were averaged from five separate runs of the experiment. This is done to reduce the system effects on the results. In addition, for conducting detailed profile measurements of the simulation performance, we have employed a commercial program execution profiling tool (QUANTIFY, from Rational Software [38]). Since monitoring the performance of an executable can be an intrusive activity, we needed a profiling tool that can be adjusted for the level of intrusiveness. For most of these experiments, only specific portions of the executable code have been profiled. For experiments on WARPED, the kernel was configured to use an aggressive cancellation strategy [37] and Least Time Stamp First (LTSF) scheduling (of objects on a single LP). The communication interface used was the Message Passing Interface (MPI) [21].

In Section 4, the set of performance-critical parameters were described. In this section, the effect of several of these parameters on the performance of synthetic models are explored. In addition to illustrating the use of the framework in exploring performance regions of the simulation environment, these experiments provide insight into the behavior of WARPED and other PDES simulators.

The first experiment involved using WSL to represent the *Ping-Pong* models developed by Barriga et al. [5]. The SWG was used to produce versions of these models where the number of (self-pinging) objects per each LP was scaled. The study was conducted for one and two LP configurations. Fig. 7 depicts the effect of the increasing number of objects on the execution time. The execution time increases uniformly; this is also reflected in the uniform decrease in the rate of committed events. If the

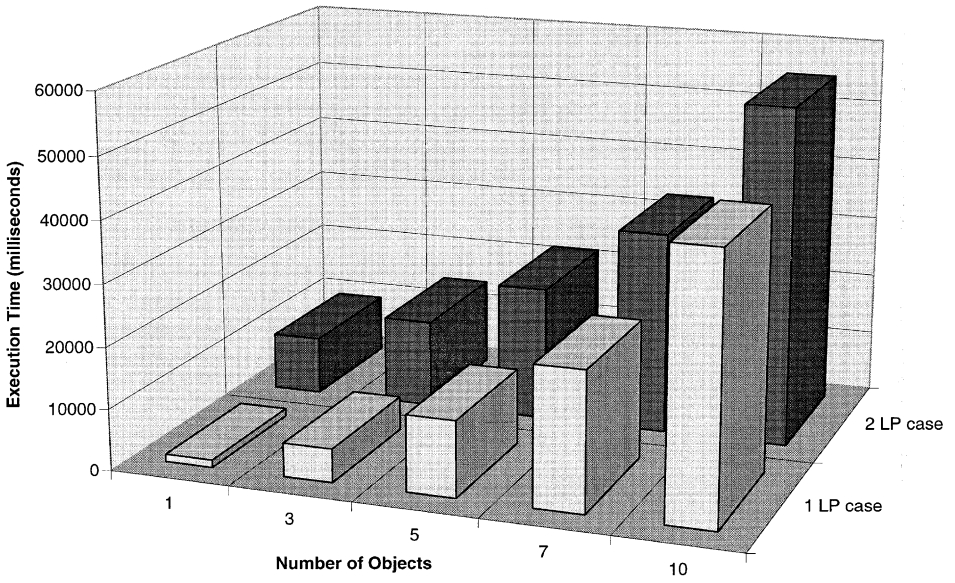

Fig. 7. Self Ping: effect on execution time.

model is executed on a single processor and the amount of work done by each LP is constant, then by varying the number of objects the overhead for scheduling can be estimated.

Upon conducting a more detailed profiling of the results, we discovered that most of the time is spent in the insertion of the event into the input queue of the object. Other overheads that affect the execution time are introduced by co-operative tasks, like GVT calculation in Time Warp and contention on internal data structures shared by the kernel. Since there is no communication between the LPs, communication overheads need not be considered – considerably simplifying the analysis. This result is intriguing because it illustrates a problem that is of considerable interest to the PDES community. The search for an efficient event list data structure has been recognized as a potential bottleneck [6,11,13,25,34,41]. Steinman [42] explored several event list data structures and compared their characteristics. However, a probability distribution was used to model the arrival of events (i.e., the event stream) and the reported observations were based on these synthetic event streams. The synthetic event stream generated by probability distributions might not reflect the event streams of real applications. The synthetic benchmarks generated by the SWG are representative of the actual model and event traces from the synthetic benchmarks are more representative of the real applications than probabilistic event streams. Hence a simulation model (synthetically generated or a mixture of real and synthetic) can be used to explore the efficiency of the different event list data structures more realistically.

In the next experiment, the effect of increasing the number of simulation processes on overall performance was investigated. The Ping model (synthetic version) was used in this experiment. The model was executed on a single processor (no communication costs). A two object experiment was constructed, and the number of LPs were scaled. Figs. 8 and 9 illustrate the effect of scaling the number of LPs on the execution time and the event commitment rate. Since the available parallelism in the model is two-way, using more than two physical processors does not yield improvement in the commitment rate. For models where the level of parallelism is not directly discernible, such an experiment can be used to find the average available parallelism. [3]

The synthetic *Ping-Pong* model was used to study the effect of event sizes on the simulation system. This study is useful because, in this benchmark the events are continuously being circulated through the communication medium. Thus, the ability of the communication medium to exchange the messages with the application layer is investigated. Fig. 10 shows that the event sizes less than 100 bytes do not perturb the system, while larger event sizes have a pronounced effect on the performance. The decrease in event commitment rate can be attributed to the saturation of the message buffers. The longer messages require a longer message propagation time as well [21].

In a related experiment, the synthetic *Ping* benchmark is used to explore the *local* buffer sizes in the message passing layer. The local buffer sizes are used

[3] This is possible as long as the limitations of the uniprocessor is not reached.

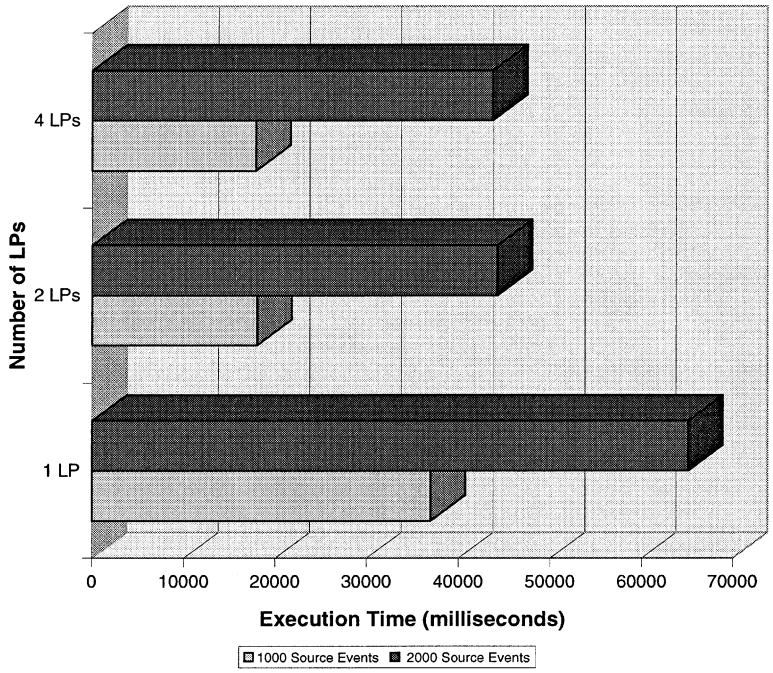

Fig. 8. Effect on execution time.

to temporarily store the messages arriving at each LP. Thus, by scaling the message generation rate at the LPs, the message receive rates of the receiving LPs are scaled and the capacity of the buffers is tested. Fig. 11 shows that the processing rates stabilize after a certain message generation rate. This can be attributed to the local buffers getting saturated after a specific message generation rate (see Fig. 12).

As the aforementioned experiments have shown, the synthetic benchmarks can be used for a wide variety of purposes. One of the best uses of these benchmarks is to study scalability of parallel discrete event simulators. The input to the synthetic workload generator can be modified to generate an assortment of benchmarks to characterize the performance of parallel discrete event simulators across several dimensions. Fig. 13 illustrates the input description to the synthetic workload generator for generating a sample synthetic benchmark. As can be seen from the figure, the example contains several parameters that can be used to study scalability. Some of these parameters are as follows:

- Number of Simulation Objects.
- Number of Logical Processes.
- Granularity of each Object.
- Event Size.
- State Size.
- File I/O.

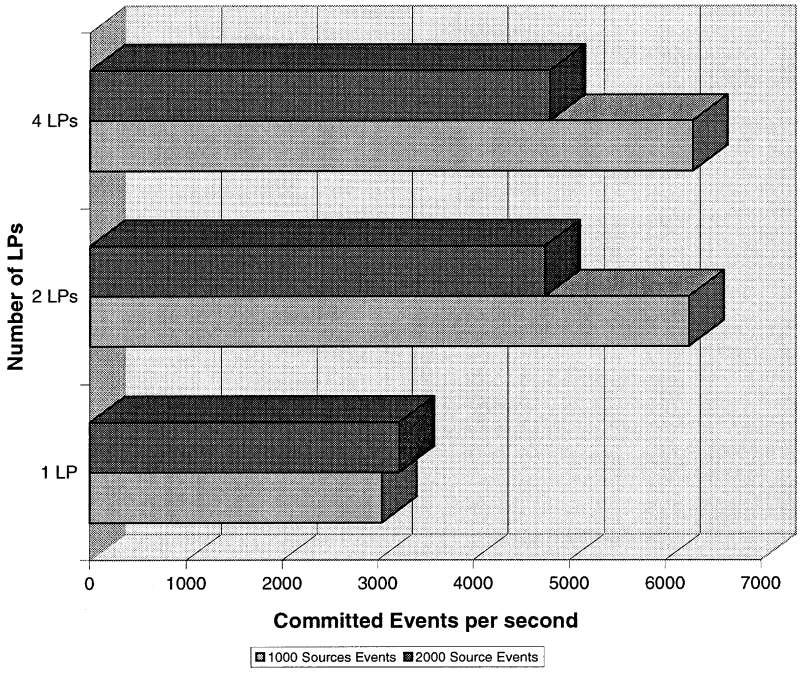

Fig. 9. Effect on commitment rate.

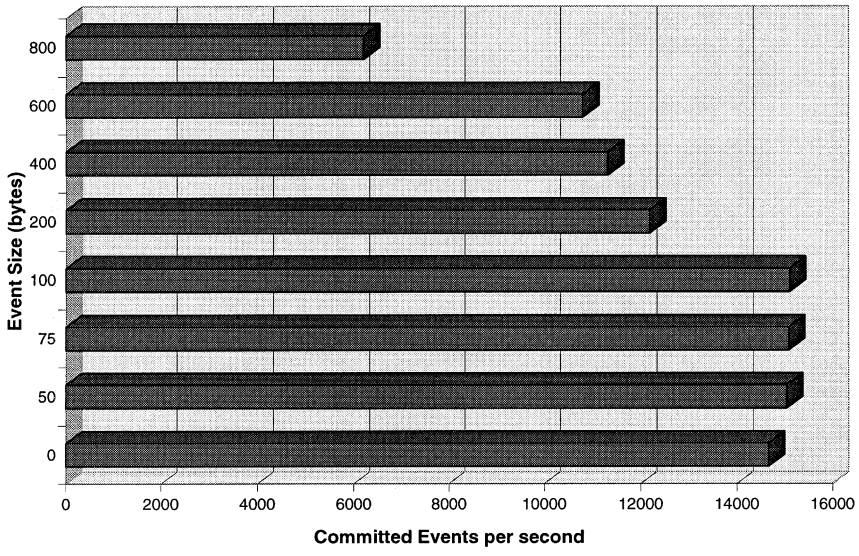

Fig. 10. Ping-Pong: effect on commitment rate.

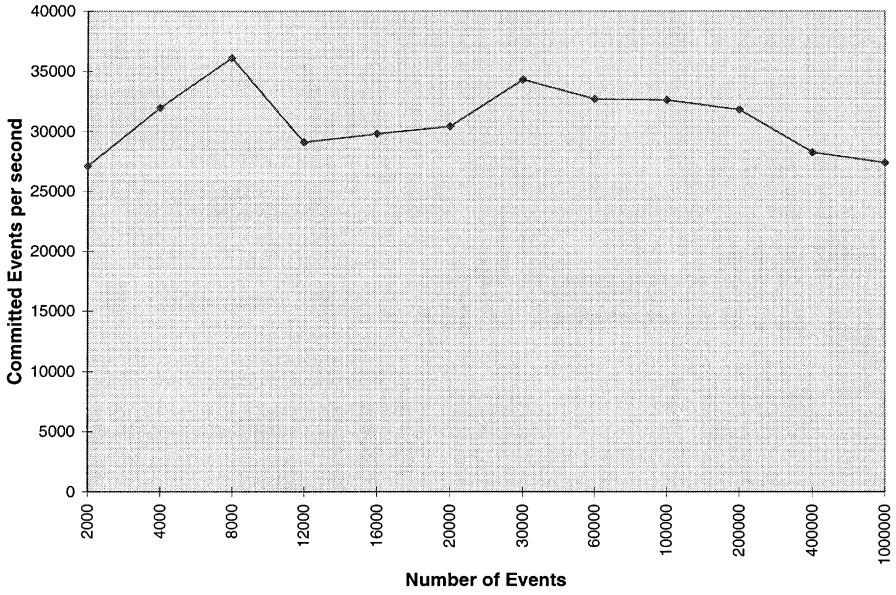

Fig. 11. Ping: effect on commitment rate.

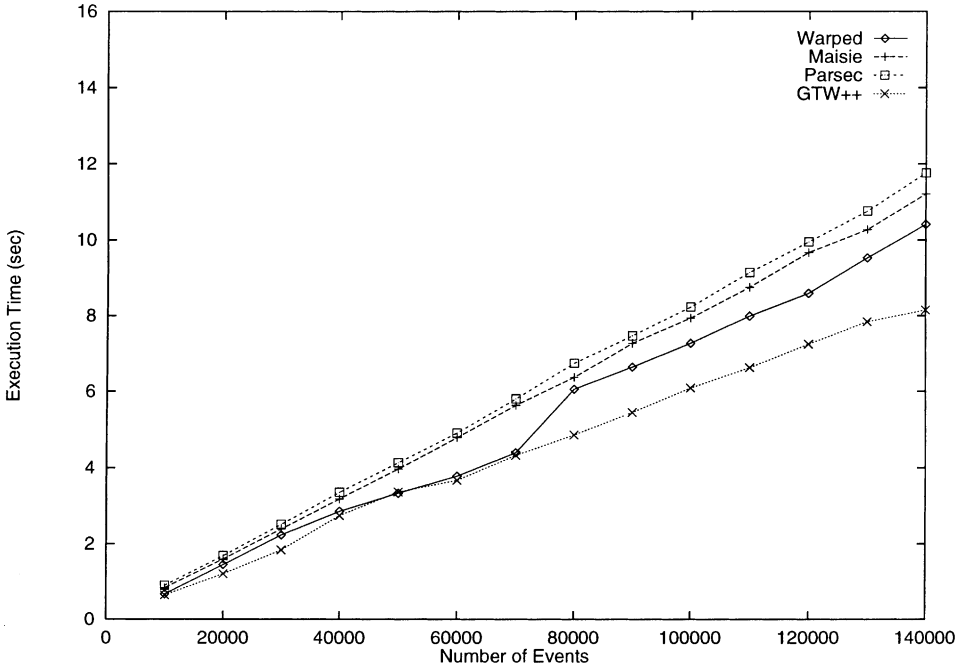

Fig. 12. Sequential execution times for the Ping-Pong example.

```
// ********************************************************************** //
// This file defines all the input parameters for the generated workload  //
// ********************************************************************** //

WORKLOAD PARAMETERS

No. of Simulation_Objects := 1000
No. of Source_Objects := 100
No. of Sink_Objects := 100
No. of Source Iteration_Count := 1000
% of Objects Performing_IO := 10
No. of LogicalProcesses := 10

TOPOLOGY PARAMETERS

No_Of_Edges := 100
Graph_Type := GRID
Graph_Display := No

SIMULATION OBJECT PARAMETERS

Average_Event_Size := 5
Event_Size_Variance := 1
Average_State_Size := 10
State_Size_Variance := 2
Average_Granularity := .01
Granularity_Variance := .05
Number of Bytes_to_Write := 10
Write_Variance := 5
Number of Bytes_to_Read := 10
Read_Variance := 5

ENDOFFILE
```

Fig. 13. A sample SWG input file.

6. Retargetting the translator

To illustrate the portability of PSAF to other simulators, translators were developed for MAISIE, PARSEC [2], GTW++ [14] and WARPED [36]. For this effort, two graduate students (who were not familiar with PSAF) were instructed to implement the translators. Approximately two days were spent in studying the interface of the different simulation environments. Once the students had familiarized themselves with PSAF and the interfaces of the target simulators, they implemented the translators in approximately two more days (for each distinct backend target). Fig. 14 shows the actual translated source code for the different simulators from a small WSL description of the Ping-Pong example. The PSAF framework along with the translators are freely available at http://www.ececs.uc.edu/~paw/psaf/.

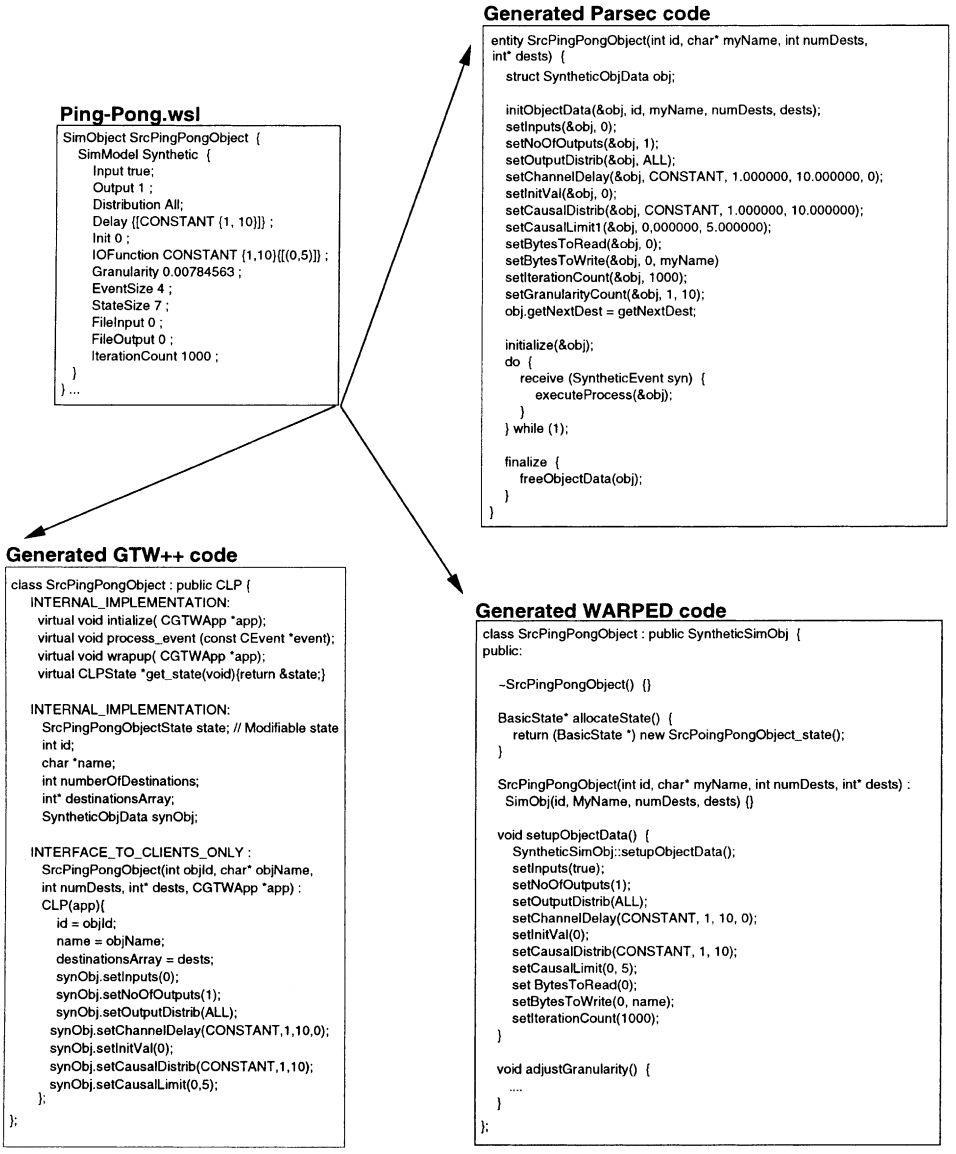

Fig. 14. Translating WSL to GTW++, PARSEC and WARPED.

As the parallel conservative and optimistic kernels of MAISIE and PARSEC were not available, we could only compare the sequential versions of MAISIE and PARSEC to other simulators. Fig. 12 presents a comparison between the execution performance of MAISIE, PARSEC, GTW++ and WARPED for the synthetic *Ping-Pong* model. From the figure, it is seen that while the sequential version of GTW++ scales well, the sequential versions of WARPED, MAISIE and PARSEC are slower. An

investigation into this result, revealed that the choice of event list data structures used in the different simulators was the major difference in the implementations. The sequential version of GTW++ uses either the (default) Calendar Queue [6] or the SkewHeap [25] as its event list data structure while WARPED (as well as MAISIE and PARSEC) use a straightforward, but less efficient linked list for their event list data structure.

The experiments discussed in this paper are intended as a sample of the capabilities of the framework. They are not sufficient for a full characterization of a PDES simulator; we are only making it possible to develop a real benchmark suite for PDES, and not actually developing such a suite. Using SWG, workloads for testing any of the performance parameters can be generated effortlessly. For example, a set of experiments to investigate the efficiency of different check-pointing algorithms can be constructed using the SWG. [4]

7. Conformance analysis

Section 5 illustrated the utility of the synthetic performance and scalability analysis framework. However, one question remains to be answered: how representative is a synthetic model of its real counterpart (i.e., how successful is the set of synthetic parameters in capturing the characteristics of the real model?).

In order to verify that synthetic representations of real workloads reflect the behavior of the original workloads, the behavior of synthetic models built for two distinct applications are compared to behavior of the real applications. The two chosen applications are: (i) the c17 benchmark from the ISCAS'85 benchmark suite; a model with low event granularity, and (ii) the Colliding Pucks model; a model with high event granularity. Fig. 15 compares the event commitment rates of the c17 gate level logic circuit and its synthetic equivalent. For a small number of input vectors, the real and the synthetic models behave similarly but as the number of input vectors increase, the real model executes faster and shows a higher processing rate than the synthetic model. This can be explained by the fact that the synthetic workloads could not match the low granularity of the gate level circuit description due to the inherent minimum weight of the statistical distributions. Conversely, examining Fig. 16 (which compares the event commitment rates of the colliding pucks model and its synthetic equivalent), the synthetic model's behavior is almost identical to the actual model's behavior for this high granularity model – increasing the confidence in the representativeness of our framework.

[4] In optimistic protocols, like Time Warp, there is a need to maintain copies of old LP states for recovery from over-optimism (*check-pointing*). The effect of the state size on the performance can be estimated by generating workloads with different values for the state size parameter.

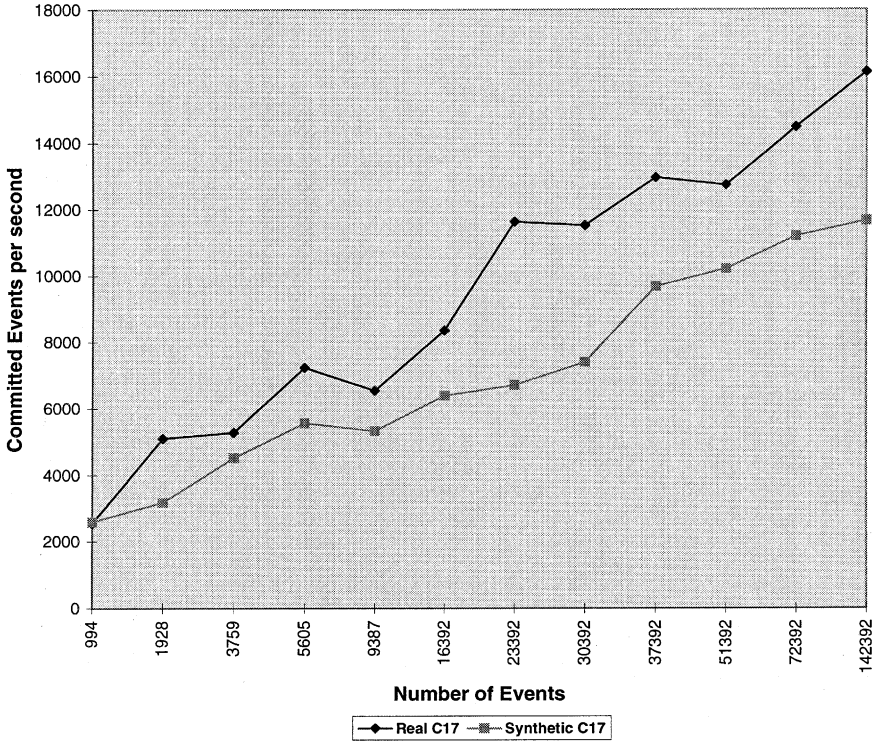

Fig. 15. C17 – real vs synthetic.

8. Conclusion

This paper introduces a PSAF for parallel discrete event simulators. One of the primary reasons restricting the development of PDES technology has been the lack of a comprehensive set of models that can be used to explore the performance of simulators objectively, and to expose bottlenecks and areas of inefficiency. PSAF addresses this need by providing a medium for the generation of models with specific performance properties, allowing controlled exploration of the performance space of the simulation environment.

PSAF is based on a WSL that allows the workload characteristics of real applications to be captured effectively using a set of performance-critical parameters. The description is then translated into simulator-specific synthetic models with equivalent performance characteristics. The effect of the performance parameters can then be studied by generating models where one or more of the parameters are modified to explore a specific class of models. This processes is facilitated using a SWG, that generates models that cover the specified ranges of the parameters under study.

In this paper, we have illustrated how the PSAF framework can be used to compare the performance of different simulators. Specifically, we compared the performance of MAISIE, PARSEC [2], GTW++ [14] and WARPED [36]. The portability of

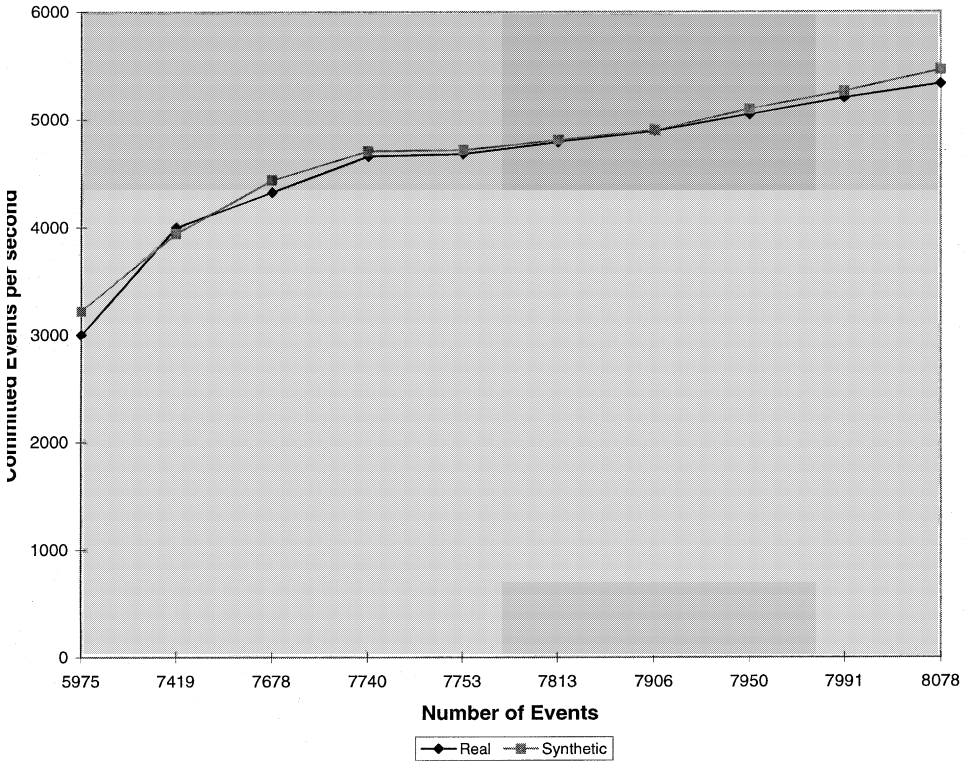

Fig. 16. Colliding pucks – real vs synthetic.

the framework was established by the quick development of translators for each of the simulators. In addition, PSAF has many important applications throughout the development cycle of simulators and models. Because PDES is used to simulate increasingly complex applications, it is important to be able to evaluate the feasibility of an implementation before embarking on a complex modeling effort. For example, since the framework supports a mixture of real and synthetic objects it can be used to build a prototype of the simulation model before the actual one is built. Moreover, the WSL representation of real models exposes their implementation details. This facilitates straightforward and accurate translation of a model to other systems such that unbiased performance comparisons are possible.

With the help of the SWG, PSAF can be used to characterize newly built simulation kernels, or the effect of new optimizations to existing kernels. Scalable workload models can be easily generated to study how well the simulation kernels scale. For each simulator, a small amount of effort is required to build the simulator-specific portion of the WSL translator. The synthetic representation permits the analysis and characterization of the performance trends of a simulator using the SWG. Moreover, SWG can be used to perform capacitance testing on the simulator. The real

models can be used, in conjunction with the synthetic models generated by SWG, to provide a comprehensive, realistic, and portable benchmark suite.

Once the benchmark suite is developed, the reporting of performance results of simulators can be standardized. A by-product of using the framework has been the development of a standard template [3] for reporting results. The template is partially generated from the WSL parameter specification file. Standardized result reporting has several advantages including the ease of interpretation, ease of comparison, and detection of unreported parameters.

References

[1] G. Amdahl, Validity of the single processor approach to chieving large scale computing capabilities, in: Proceedings of the AFIPS Spring Joint Computer Conference, April 1967, pp. 483–485.
[2] R.L. Bagrodia, W. Liao, Maisie: a language for the design of efficient discrete-event simulations, IEEE Transactions on Software Engineering 20 (4) (1994) 225–238.
[3] V. Balakrishnan, A framework forperformance evaluation of parallel discrete event simulators, Master's Thesis, University of Cincinnati, Cincinnati, OH, 1997.
[4] V. Balakrishnan, P. Frey, N. Abu-Ghazaleh, P.A. Wilsey, A framework for performance analysis of parallel discrete event simulators, in: Proceedings of the 1997 Winter Simulation Conference, December 1997.
[5] L. Barriga, R., Ronngren, R. Ayani, Benchmarking parallel simulation algorithms, in: Proceedings of the IEEE 1st International Conference on Algorithms and Architectures for Parallel Processing, 1995, pp. 611–620.
[6] R. Brown, Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem, Communications of the ACM 31 (10) (1988) 1220–1227.
[7] R.E. Bryant, Simulation on a distributed system, in: Proceedings of the 1st International Conference on Distributed Computing Systems, Washington, DC, 1979, IEEE Computer Society Press, Silver Spring, MD, pp. 544–552.
[8] CAD Benchmarking Lab, NCSU, ISCAS'89 Benchmark Information (available at http://www.cbl.ncsu.edu/www/CBL_Docs/iscas89.html).
[9] K.M. Chandy, J. Misra, Distributed simulation: a case study in design and verification of distributed programs, IEEE Transactions on Software Engineering 5 (5) (1979) 440–452.
[10] K.M. Chandy, J. Misra, Asynchronous distributed simulation via a sequence of parallel computations, Communications of the ACM 24 (11) (1981) 198–206.
[11] S. Chung, V. Rego, A performance comparison of event calendar algorithms: an empirical approach, Software Practice and Experience 23 (10) (1993) 1107–1138.
[12] D. Conklin, J. Cleary, B. Unger, The sharks world: a study in distributed simulation design, in: Distributed Simulation, SCS Simulation Series, 1990, pp. 157–160.
[13] C. Crane, Linear lists and priority queues as balanced binary trees, Tech. Rep. Tech. Rept. STAN-CS-72-259, Computer Science Department, Stanford University, 1972.
[14] S. Das, R. Fujimoto, K. Panesar, D. Allison, M. Hybinette, GTW: a Time Warp system for shared memory multiprocessors, in: J.D. Tew, S. Manivannan, D.A. Sadowski, A.F. Seila (Eds.), Proceedings of the 1994 Winter Simulation Conference, December 1994, pp. 1332–1339.
[15] A. Ferscha, J. Johnson, A testbed for parallel simulation performance predictions, in: 1996 Winter Simulation Conference Proceedings, December 1996.
[16] R. Fujimoto, Performance measurements of distributed simulation strategies, Tech. Rep. UU-CS-TR-87-026a, University Of Utah, Salt Lake City, November 1987.
[17] R. Fujimoto, Parallel discrete event simulation, Communications of the ACM 33 (10) (1990) 30–53.
[18] R. Fujimoto, Performance of Time Warp under synthetic workloads, in: Proceedings of the SCS Multiconference on Distributed Simulation, vol. 22 (1), 1990, pp. 23–28.

[19] R. Fujimoto, Parallel and distributed discrete event simulation: algorithms and applications, in: Proceedings of the 1993 Winter Simulation Conference, 1993, pp. 106–114.
[20] J.B. Gilmer Jr., An assessment of Time Warp parallel discrete event simulation algorithm performance, in: Distributed Simulation, SCS Simulation Series, 1998, pp. 45–49.
[21] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1994.
[22] P. Hontallas, et al., Performance ofcolliding pucks simulation on the Time Warp operating system, in: Distributed Simulation, Society of Computer Simulation, 1989, pp. 3–7.
[23] D. Jefferson, Virtual time, ACM Transactions on Programming Languages and Systems 7 (3) (1985) 405–425.
[24] J.L. Gustafson, G.R. Montry, R.E. Benner, Development of parallel methods for a 1024-node hypercube, SIAM Journal on Scientific and Statistical Computing 9 (4) (1998) 609–638.
[25] D. Jones, An empirical comparison of priority-queue and event-set implementations, Communications of the ACM 29 (4) (1986) 300–311.
[26] D.L. Kiskis, Generation of synthetic workloads for distributed real-time computing systems, Ph.D. Thesis, University of Michigan, 1992.
[27] D. Kumar, An approximate method to predict the performance of a distributed simulation scheme, in: Proceedings of the 1989 Internationl Conference on Parallel Processing, August 1989, pp. 259–262.
[28] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565.
[29] Y.-B. Lin, E.D. Lazowska, J.-L. Baer, Conservative parallel simulation for systems with no lookahead prediction, in: Distributed Simulation, Society for Computer Simulation, January 1990, pp. 144–149.
[30] K. Mehlhorn, S. Näher, C. Uhrig, The LEDA User Manual Version R 3.4.1, 1996.
[31] J. Misra, Distributed discrete-event simulation, Computing Surveys 18 (1) (1986) 39–65.
[32] W.E. Nagel, M.A. Linn, Benchmarking parallel programs in a multiprogramming environment: The PAR-bench system, Parallel Computing 17 (1991) 1303–1321.
[33] T.J. Parr, Language Translation Using PCCTS and C++, Automata Publishing Company, January 1997.
[34] S. Prasad, B. Naquib, Effectiveness of global event queues in rollback reduction and loadbalancing, in: Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS 95), 1995, pp. 187–190.
[35] B.R. Preiss, The Yaddes distributed discrete event simulation specification language and execution environments, in: Proceedings of the SCS Multiconference on Distributed Simulation, 1989, pp. 139–144.
[36] R. Radhakrishnan, D.E. Martin, M. Chetlur, D.M. Rao, P.A. Wilsey, An object-oriented Time Warp simulation kernel, in: D. Caromel, R.R. Oldehoeft, M. Tholburn (Eds.), Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98), Springer, Berlin, 1998, pp. 13–23.
[37] R. Rajan, R. Radhakrishnan, P.A. Wilsey, Dynamic cancellation: selecting Time Warp cancellation strategies at runtime, VLSI Design 9 (3) (1999) 237–251.
[38] Rational Software Corporation,18880 Homestead Rd., Cupertino, CA 95014. Quantify Runtime Analysis Tool (available at http://www.rational.com/products/pqc/index.jsp).
[39] P.F. Reynolds Jr., P. Dickens, SPECTRUM: a parallel simulation testbed, in: Hypercube Conference, 1989.
[40] B. Samadi, Distributed simulation, algorithms and performance analysis, Ph.D. Thesis, Computer Science Department, University of California, Los Angeles, CA, 1985.
[41] D. Sleator, R. Tarjan, Self adjusting binary search trees, Journal of the ACM 32 (3) (1985) 652–686.
[42] D. Steinman, Discrete-event simulation and the event horizon part 2: Event list management, in: Proceedings of the 10th Workshop on Parallel and Distributed Simulation (PADS96), 1996, pp. 170–178.
[43] X.-H. Sun, J.L. Gustafson, Towards a parallel performance metric, Parallel Computing 17 (1991) 1093–1109.