Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

A FRAMEWORK FOR PERFORMANCE ANALYSIS OF PARALLEL DISCRETE EVENT SIMULATORS

Vijay Balakrishnan
Peter Frey
Nael B. Abu-Ghazaleh
Philip A. Wilsey

Computer Architecture Design Laboratory
P.O. Box 210030, Cincinnati, Ohio 45221–0030, U.S.A.

ABSTRACT

A framework for performance analysis of parallel discrete event simulators is presented. The centerpiece of this framework is a platform-independent *Workload Specification Language* (WSL). WSL is a language that allows the characterization of simulation models using a set of fundamental performance-critical parameters. WSL also implements a facility for representing real models. For each simulator to be tested, a WSL translator is used to generate synthetic platform-specific simulation models that conform to the performance characteristics captured by the WSL description. Accordingly, sets of portable simulation models that explore the effects of the different parameters, individually or collectively, on the performance can be constructed. The construction of the workload simulation models is assisted using a *Synthetic Workload Generator* (SWG). The utility of the system is demonstrated with the generation of a representative set of experiments. The described framework can be used to create a standard benchmark suite that consists of a mixture of real simulation models, selected from different application domains, and synthetic models generated by SWG.

1 INTRODUCTION

Performance analysis of parallel discrete event simulators is a task complicated by the large number of interrelated factors affecting performance. (Throughout this paper, we use the term *performance* to mean the inverse of execution time.) Accurate and unbiased performance analysis is important throughout the development cycle of simulators and models. Traditionally, *Speedup* has been used as a relative performance indicator. Since speedup is defined with respect to *a* sequential simulator (left to the taste of the tester), it is difficult to get directly comparable speedup figures. This is especially true if the models under comparison are not identical. Moreover, while speedup is a useful measure for coarsely comparing the performance of systems, it does not expose sufficient detail about the factors leading to this particular speedup figure.

This paper presents a framework for performance analysis of parallel discrete-event simulators. The framework 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 (using a simulator-specific translator) to different simulation back-ends. A portable automated WSL translator, where only minimal back-end description is needed for each simulator, is provided in order to simplify the task of building simulator-specific translators (Balakrishnan 1997). Thus, synthetic workloads with tunable performance-related parameters are generated, allowing the performance of the simulator to be investigated under controlled workload conditions. The analysis can be further aided by using a *Synthetic Workload Generator* (SWG); a program that automatically generates WSL descriptions that test the effect of subsets of the model parameters on the performance. The portability of the synthetic models afforded by the system allows unbiased and thorough comparison of simulators.

The ultimate goal of this project is to provide a standard benchmark suite that studies the performance space of the simulators using realistic models. To that end, WSL implements a feature for representing real models directly. The mixture of real models, representative of different simulation domains, serves alongside the synthetic models generated by the SWG to provide a starting point towards the standard benchmark suite.

The remainder of this paper is organized as follows. Section 2 describes *Parallel Discrete-Event Simulation* (PDES). Section 3 discusses some related benchmarking efforts. Section 4 presents the performance analysis framework, and discusses its various components. Section 5 presents an example of the use of the framework by developing a set of experiments for a PDES and analyzing the performance. Finally, Section 6 presents some concluding remarks.

2 BACKGROUND — PDES

In this section, we briefly overview PDES. In a parallel discrete event simulation, 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 (Lamport 1978; Jefferson 1985). There are two basic synchronization protocols used to ensure that this condition is not violated: (i) *conservative* and (ii) *optimistic*. Conservative protocols (Misra 1986; Chandy and Misra 1981) strictly avoid causality errors, while optimistic ones (Jefferson 1985; Fujimoto 1990) allow causality errors to occur, but implement a *rollback* mechanism enabling recovery.

3 RELATED WORK

There are several empirical and analytical studies (Fujimoto 1987; Fujimoto 90; Barriga, Ronngren, and Ayani 1995; Samadi 1985; Kumar 1989) of the performance of PDES algorithms. Most of these studies appear in the context of evaluating the impact a particular simulator improvement on the performance. The utility of a new simulator protocol/optimization cannot assessed accurately using such a narrow comparison. For example, the simulation model used for the comparison greatly influences the performance. Furthermore, the implementation used for operations such as memory allocation, event-list management, GVT calculation and deadlock detection affect the performance. The number of factors influening performance is large and, therefore, a set of benchmarks from the PDES application domain (Fujimoto 1993) is needed to sufficiently characterize the performance of a simulator under different working conditions. Moreover, this benchmark suite must be independent of the simulator to allow unbiased comparison of simulators. Only a small number of the models described by Fujimoto (Fujimoto 1993) are available freely. Moreover, some models that are available have several different implementations.

Several efforts have attempted to build useful performance analysis frameworks for PDES simulators. The suggested frameworks allow the user to select a simulation configuration from a set of basic blocks supported in the framework. (In the context of performance analysis frameworks, *user* primarily refers to a simulator developer.) The user builds an application and executes it under the different simulator configurations to analyze the effect of the parameters. For example, Reynolds's (Reynolds 1989; Reynolds Jr 1989; Reynolds and Dickens 1989) SPECTRUM testbed allows a user to implement a simulator configuration (protocol), supply an application, and specify some of the key parameters. The performance parameters incorporated in the framework include determinism, queuing, processing delays, causality, state change characteristics, balance, activity, and connectivity. Thus, the effect of these parameters on performance can be studied. The major drawback of this framework is the need to re-develop the simulation kernel using the testbed for each configuration. Furthermore, the application cannot be tested on any other simulator. Gilmer (Gilmer Jr 1988) also defines some parameters and uses them to build simulation models.

Ferscha (Ferscha and Johnson 1996) develops a tool for performance prediction of Time Warp (Jefferson 1985) 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, testbeds that are currently in use include Yaddes (Preiss 1989) and Maise (Bagrodia, Chandy, and Toh 1991). 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.

4 THE FRAMEWORK

This section discusses the different components of the performance analysis framework suggested in this paper. An overview of the framework is shown in Figure 1. Central to the framework is the *Workload Specification language* (WSL), a language for capturing the performance-critical attributes of applications. Since the performance-critical attributes are explicitly visible, WSL can be used to generate syn-

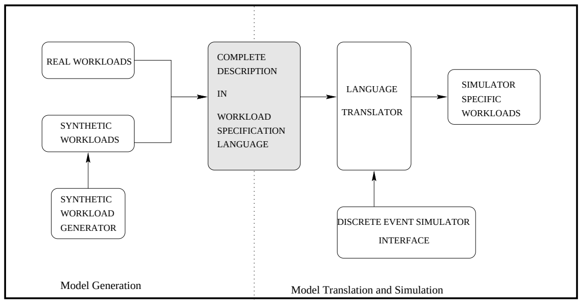
REAL WORKLOADS
SYNTHETIC WORKLOADS
SYNTHETIC WORKLOAD GENERATOR
COMPLETE DESCRIPTION IN WORKLOAD SPECIFICATION LANGUAGE
LANGUAGE TRANSLATOR
SIMULATOR SPECIFIC WORKLOADS
DISCRETE EVENT SIMULATOR INTERFACE
Model Generation
Model Translation and Simulation

Figure 1: The Performance Analysis Framework

thetic workloads using a *Synthetic Workload Generator* (SWG). Synthetic workload generation allows the performance space of the simulator to be explored methodically. Moreover, real models can be represented using WSL. An automatic translator, at the back-end of the framework, translates WSL descriptions to equivalent synthetic models that conform to the performance characteristics specified by the WSL description; real models must be ported manually.

4.1 Workload Specification Language

This section describes the Workload Specification Language (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 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 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 among simulation languages with high accuracy, allowing an impartial comparison to be carried out. In addition, synthetic workload descriptions, written using WSL, can be directly applied to the different simulators (using the automatic translators).

A workload specification in WSL consists of a list of simulation object definitions (**SimObject**) followed by a list of instantiations of these objects. Every simulation object may have an unlimited number of definitions. These definitions correspond to the different simulation kernels or to a synthetic (**Synthetic**) representation. The translator generates translated code depending on the target simulation platform. In the remainder of this section, we overview the representation and translation facets of WSL.

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. The synthetic object is used to build a synthetic workload. Ideally, the characteristic of the synthetic workloads should match that of a real workload. For this reason, a workload characterization of several PDES models was carried out and parameters that affect the performance were isolated. Several of the parameters have been identified previously by Fujimoto (Fujimoto 90), and Reynolds (Reynolds 1989). Some of the important parameters supported by WSL are:

1. Computation Granularity
2. Memory Requirement
3. Topology
4. Input and Output Behavior
5. Event Population
6. Event Probability
7. Event Delay
8. Number of Processors
9. Number of Physical Processes
10. Ease of Verification
11. Initial configuration

The last two parameters do not relate to performance, but have been included to provide information that can be used to verify the correctness of the simulation. A complete breakdown of the parameters is not provided here for brevity (Balakrishnan 1997). The synthetic description given in figure 2 demonstrates how the parameters are specified. A complete model consists of a set of such objects with an associated connectivity pattern (topology).

4.1.2 Real-Model Representation

WSL also supports real model descriptions, inserted as simulation-specific code. Each simulation environment is assigned a unique tag which is used to mark model descriptions suitable for it (using the keyword `SimModel`). When generating simulation models for a given simulation environment, the translator searches for the tag corresponding to it. The structure of a real object description is shown in figure 3. Every section is optional; the structure serves as a guideline. The design was found to be well suited for writing the description in C++. However, it should also be useful for other languages because of the organization of

```
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 |
                   NORMAL | EXPONENTIAL |
                   BINOMIAL | CONSTANT}
                  { seed, seed }
  separation interval list := {x,y}
```

Figure 2: Structure of a Synthetic Object

the design. More precisely, the organization provides a clear methodology for re-writing an object description in another language, by making the primary constituents individually-available in self-contained description clauses.

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 LP assignment. Each object definition may be instantiated multiple times, and connected to other objects using the net-list. Every simulation object (`SimObject`) can 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)

The Synthetic Workload Generator (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, such that the regions of good and bad performance are identified.

```
SimObject nameOfObject {
  SimModel Warped {
    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 ]];
}}}
```

Figure 3: Structure of a Real Object

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 specified parameters. 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) (Mehlhorn, Näher, and Uhrig 1996). 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 every node to a synthetic simulation object by filling in the values for the parameters shown in Figure 2. Once the second phase terminates, the simulation object description and the net-list for the WSL description of the workload are ready.

Figure 4 shows the structure of the SWG. The pa-

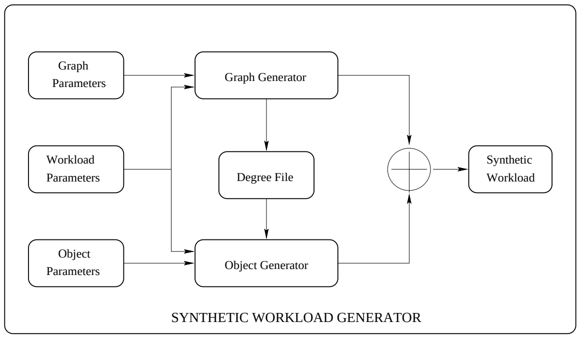

Figure 4: Structure of the SWG

rameter 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. It is designed using PCCTS (Parr 1995), a compiler construction toolkit. The translator consists of a parser that calls predefined functions in the `PublishClass`; there is one `PublishClass` for each simulation environment supported by the translator. Figure 5 shows the organization of the translator. Building a translator for a new simulation environment involves filling in the pre-defined methods for its `PublishClass`; only the back-end to the generator need be modified.

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 object to different processors as required by the WSL specification.

We have constructed a translator for WARPED, a Time-Warp synchronized simulation kernel (Martin, McBrayer, and Wilsey 1995). the complete system has been used for performance evaluation of the WARPED kernel. The framework was written in C++ and the translator implemented using PCCTS (Parr 1995), a freely available compiler construction toolkit. The system was tested using the GNU g++ compiler.

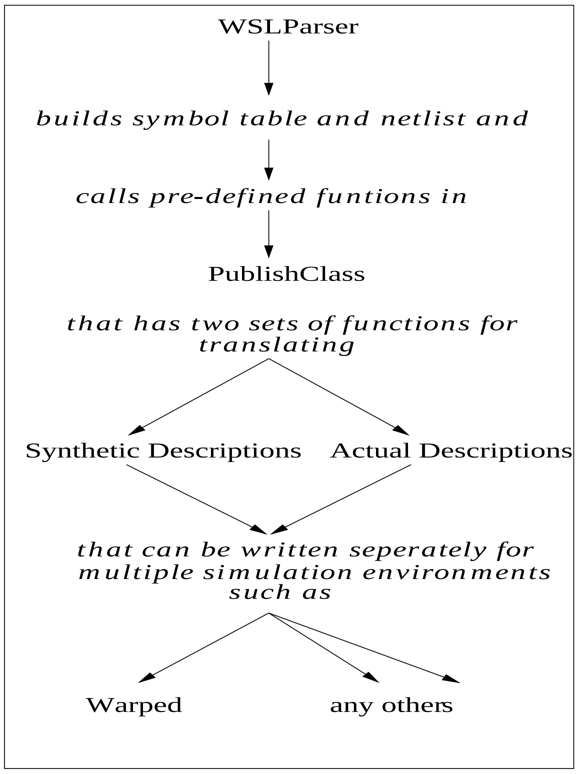
WSLParser
builds symbol table and netlist and
calls pre-defined funtions in
PublishClass
that has two sets of functions for translating
Synthetic Descriptions
Actual Descriptions
that can be written seperately for multiple simulation environments such as
Warped
any others

Figure 5: Structure of the WSL Translator

5 EXPERIMENTS

In this section, the framework is used to analyze the performance of the WARPED (Martin, McBrayer, and Wilsey 1995) simulation kernel. The models used to perform the experiments were generated using the SWG. They were then translated to WARPED code by a translator written for WARPED. The experiments were conducted on a SUN SPARCCENTER 1000 with 4 processors. The GNU g++ compiler with the optimizations (`-O2`) was used to compile the models. The WARPED kernel was used with the aggressive cancelation (Rajan and Wilsey 1995) strategy, Least Time Stamp First (LTSF) scheduling policy, and using the MPI communication implementation. The synthetic model used consisted of 100 simulation objects that were randomly partitioned among two LPs. The topology used was a GRID.

The effect of increasing event granularity is investigated first. The results are shown in the Figure 6. Higher granularity does not affect efficiency (ratio of committed events to the total number processed events). Event granularity is independent of the scheduling time of each event. For very low granularities (.001 to 1 microsecond) the drop in event rates is more gradual than the rate shown in Figure

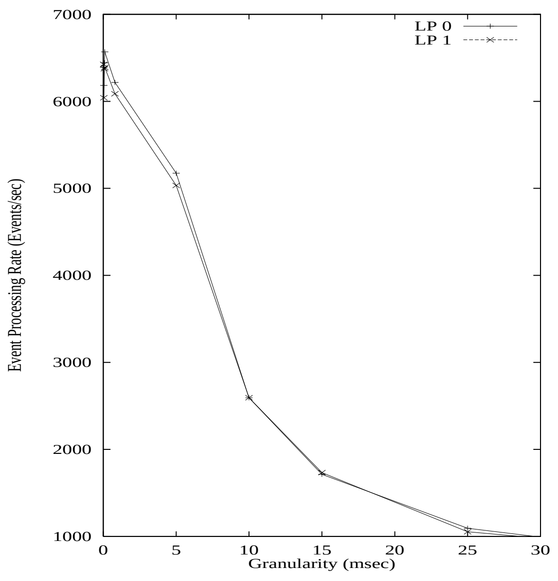

Figure 6: The Effects of Event Granularity

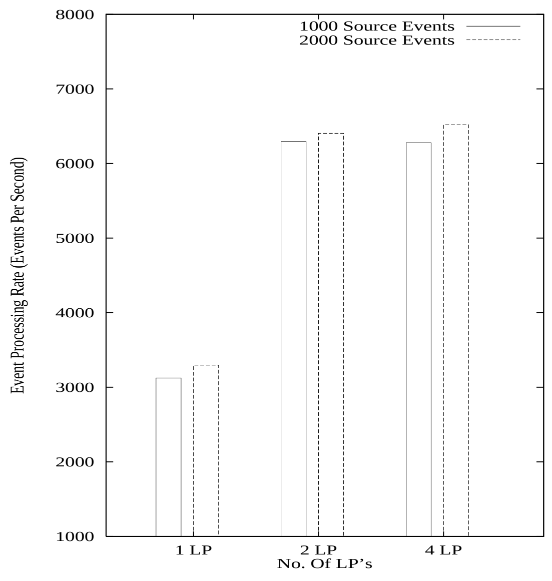

Figure 7: The Effect of Parallel Execution

6.

Since WSL provides support for partitioning, the same model was studied using two and four LPs; the 1 LP case serves as the base case. The granularity value used for this study is 0.1 microsecond. The results are shown in Figure 7.

For the next experiment, WSL was used to represent the *ping-pong* models presented by Barriga (Barriga, Ronngren, and Ayani 1995). The results of scaling these benchmarks is shown in Figure 8 and Figure 9. SWG was used to produce scaled versions of these models from the synthetic equivalent description of the real model. Figure 8 shows the effect of increasing messages sizes on the event processing rates. The Ping model (Figure 9) is used to assess the effect of the buffer sizes in the message passing layer. After the buffer becomes full, the event processing rates stabilize. This occurs after a momentary decrease in event processing rate (caused by the full buffers).

The experiments discussed in this section are intended as a sample of the capabilities of the framework; they are not sufficient for a full characterization of WARPED. Using SWG, workloads for testing any of the performance parameters can be generated easily. Thus, similar experiments can be carried out to study the effects of other parameters on the performance.

Finally, it is important to verify that synthetic representations of real models reflect the behavior of the original models. In order to verify this correspondence, we evaluated the behavior of synthetic models for some circuits of the ISCAS'85 (CAD Benchmarking Lab, NCSU 1989) benchmark suite relative to the behavior of the real models. The results are encouraging; however, more extensive evaluations are necessary to strengthen this conclusion (Balakrishnan 1997).

6 CONCLUSIONS

The framework presented in this paper introduces a common and uniform methodology for performance analysis and benchmarking of simulation environments. The framework, which is based on a Workload Specification Language (WSL), provides a simple platform for capturing workload characteristics and translating the workload description into synthetic models with equivalent performance characteristics. The performance analysis framework 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 a Synthetic Workload Generator

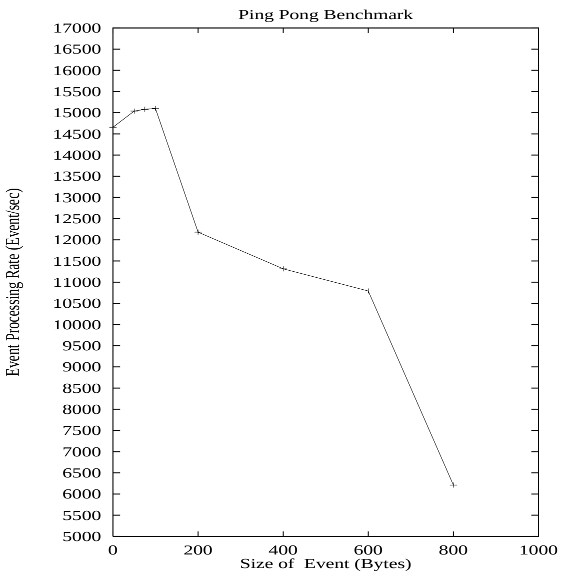

Figure 8: The Effects of Event Size

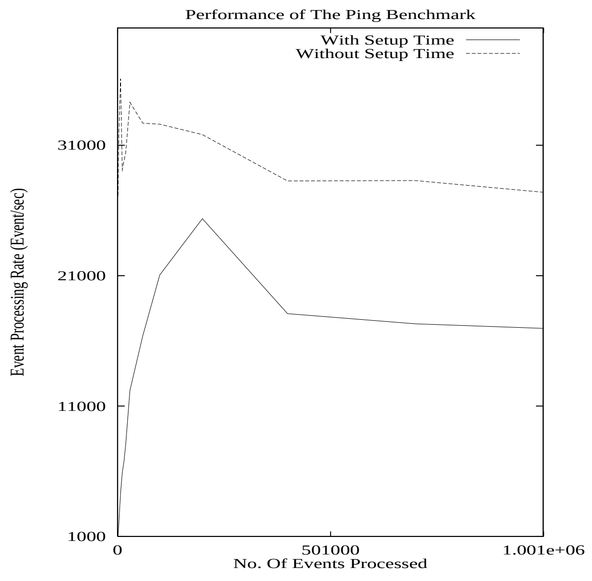

Figure 9: The Effect of Event Population

(SWG), the framework can be used to characterize newly built simulation kernels, or the effect of new optimizations on existing kernels. For each simulator, a small amount of effort is required to write the back-end for 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, alongside the synthetic models generated by SWG, to provide a comprehensive, realistic, and portable benchmark suite.

The benchmark suite is especially useful if it is continuously augmented with models from emerging application domains — continuing to reflect realistic workloads. We have started assembling a benchmark suite of real models that will be complemented by SWG produced workloads. The suite currently includes an implementation of gate-level digital logic simulator, a sharks world model (Conklin, Cleary, and Unger 1990), and the set of ping models proposed by Barriga (Barriga, Ronngren, and Ayani 1995). The circuits used for the digital-logic simulation are from the ISCAS'89 (CAD Benchmarking Lab, NCSU 1989) benchmark suite. Additional models are being added to the benchmark suite.

ACKNOWLEDGMENT

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

REFERENCES

Bagrodia, R., K. M. Chandy, and L. W. Toh 1991. Unifying framework for distributed simulation. *ACM Transaction on Modeling and Computer Simulation 1*(4): 348–385.

Balakrishnan, V. 1997. A Framework for Performance Evaluation of Parallel Discrete Event Simulators. Master's thesis, ECE-CS Deptartment, University of Cincinnati.

Barriga, L., R. Ronngren, and R. Ayani 1995. Benchmarking parallel simulation algorithms. In *Proceedings of the IEEE 1st Intl. Conf. on Algorithms and Architectures for Parallel Processing*, 611–620.

CAD Benchmarking Lab, NCSU 1989. *ISCAS'89 and '85 Benchmark Information.* CAD Benchmarking Lab, NCSU (available at `http://www.cbl.ncsu.edu/www/CBL_Docs/`).

Chandy, K. M. and J. Misra 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM 24*(11): 198–206.

Conklin, D., J. Cleary, and B. Unger 1990. The sharks world : (A study in distributed simulation design). In *Distributed Simulation*, 157–160. SCS Simulation Series.

Ferscha, A. and J. Johnson 1996. A testbed for parallel simulation performance predictions. *Proceedings of the 1996 Winter Simulation Conference.*

Fujimoto, R. 1987. Performance measurements of distributed simulation strategies. Technical Report UU–CS–TR–87–026a, University Of Utah, Salt Lake City.

Fujimoto, R. 1990. Parallel discrete event simulation. *Communications of the ACM 33*(10): 30–53.

Fujimoto, R. 1993. Parallel and distributed discrete event simulation: Algorithms and applications. In *Proceedings of the 1993 Winter Simulation Conference*, 106–114.

Fujimoto, R. 1990. Performance of Time Warp under synthetic workloads. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 23–28. SCS.

Gilmer Jr, J. B. 1988. An assessment of Time Warp parallel discrete event simulation algorithm performance. In *Distribute Simulation*, 45–49. SCS Simulation Series.

Jefferson, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems 7*(3): 405–425.

Kumar, D. 1989. An approximate method to predict the performance of a distributed simulation scheme. In *Proceedings of the 1989 Internation Conference on Parallel Processing*, 259–262.

Lamport, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM*, 558–565.

Martin,- D. E.,- T. McBrayer,- and P. A. Wilsey 1995. WARPED:- A time warp simulation kernel for analysis and application development.- (available on- the- www- at `http://www.ece.uc.edu/~paw/warped/`).

Mehlhorn, K., S. Näher, and C. Uhrig 1996. The LEDA User Manual Version R 3.4.1.

Misra, J. 1986. Distributed discrete-event simulation. *Computing Surveys 18*(1): 39–65.

Parr, T. J. 1995. Language translation using PCCTS and C++. A Reference Guide. `http://www.parr-research.com/~parrt`.

Preiss, B. R. 1989. The Yaddes distributed discrete event simulation specification language and execution environments. In *Proceedings SCS Multiconf. on Distributed Simulation*, 139–144.

Rajan, R. and P. A. Wilsey 1995. Dynamically switching between lazy and aggressive cancellation in a time warp parallel simulator. In *Proceedings of the 28th Annual Simulation Symposium*, 22–30. IEEE Computer Society Press.

Reynolds, J. P. F. 1989. A spectrum of options for parallel simulation. In *Proceedings 1988 Winter Simulation Conference*, 325–332.

Reynolds, P. and P. Dickens 1989. SPECTRUM: A Parallel Simulation Testbed. In *Hypercube Conference*.

Reynolds Jr, P. F. 1989. Comparative analysis of parallel simulation protocols. In *Proceedings of the 1989 Winter Simulation Conference*, 671–678.

Samadi, B. 1985. *Distributed Simulation, Algorithms and Performance Analysis*. Ph. D. thesis, Computer Science Department, University of California, Los Angeles, CA.

AUTHOR BIOGRAPHIES

VIJAY BALAKRISHNAN is a graduate research assistant in the Dept of ECECS at the Univ of Cincinnati. He obtained his undergraduate engineering degree in Electronics and Communication from Birla Institute of Technology, Ranchi, India. His research interests include: Parallel Discrete Event Driven Simulation, and Performance Evaluation and Benchmarking.

PETER FREY Dipl.-Ing.(FH) is a senior Ph.D. student in the Distributed Processing Laboratory at the University of Cincinnati. His research interests include: Mixed-Mode Simulation, Parallel Discrete Event Simulation, Formal Software Specification, and Hardware Description Languages.

NAEL B. ABU-GHAZALEH is a post-doctoral research associate with the Computer Architecture Design Lab, Dept of ECECS at the University of Cincinnati. His primary reasearch interests are Massively Parallel Architectures, Parallel Discrete Event Simulation, Large Scale Interconnection Networks, and Knowlege-based Decision Systems. He is a member of the ACM, and IEEE.

PHILIP A. WILSEY is an assistant professor in the Dept. of ECECS at the Univ of Cincinnati. His research interests include: Parallel Discrete Event Driven Simulation, Hardware Description Languages, Design Automation Systems, SIMD/MIMD Multiprocessors, and Computer Architecture. Dr. Wilsey is a member of the ACM, IEEE, and DASC.