Predicting Performance Impacts due to Resolution Changes in Parallel Simulations

Dhananjai M. Rao
CSA Department
Miami University
Oxford OH 45056, USA
raodm@muohio.edu

Philip A. Wilsey
Department of ECECS
University of Cincinnati
Cincinnati, OH 45221–0030, USA

Multi-resolution models are often used to accelerate simulation-based analysis without significantly impacting the fidelity of the simulations. We have developed a web-enabled, component-based, multi-resolution modeling and Time Warp synchronized parallel simulation environment called WESE (Web-Enabled Simulation Environment). WESE uses a methodology called Dynamic Component Substitution (DCS) to enable abstractions or refinements to a given model. However, effectively utilizing abstractions, whether they are DCS-based or not, is a complex and time-consuming task. The complexity arises because not all abstractions improve simulation performance due to a myriad of factors related to model characteristics, synchronization protocol overheads and simulation-platform configuration. The overheads involved in identifying optimal model resolution have been exacerbating effective use of multi-resolution simulations, including our DCS-based approach. In an endeavor to minimize the time taken to identify performance impacts of resolution changes, this study proposes a DCS Performance Prediction Methodology (DCSPPM). It predicts simulation performance changes due to DCS transformations via static analysis of the model. Static analysis uses platform-specific performance characteristics of components constituting the model. DCSPPM yields quantitative estimates of performance impacts which are used by the modeler to select appropriate transformations. This article presents DCSPPM, its implementation in WESE and its empirical evaluation. The inferences drawn from the experiments prove that DCSPPM estimates have errors of less than 5% for a variety of models. Furthermore, DCSPPM executes orders of magnitude faster than corresponding shortest test simulations. Note that applicability of DCSPPM is not restricted to WESE but can be extended to other Time Warp synchronized simulators.

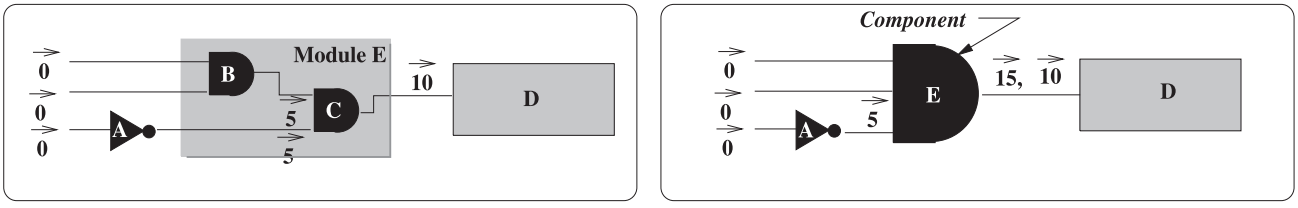
Keywords:

1. Introduction

In-depth study and analysis of modern systems such as microprocessors and communication networks is crucial to effectively design, develop, manufacture, control and maintain such systems [1, 2]. The accelerated increase in size and complexity of the systems catalyzed by the need for comprehensive knowledge furnished using intuitive representations has necessitated the use of computer-based simulations for their study and analysis. Simulation is widely used because it is an intuitive, cost-effective and non-destructive methodology for the study and analysis of a wide spectrum of systems [2, 3]. It enables exploration of complicated scenarios that would otherwise be difficult or impossible to analyze [4]. Parallel simulation techniques are often used to provide more optimal space-time tradeoffs to enable simulation of large models in reasonable timeframes [1, 5].

SIMULATION, Vol. 84, Issue 10/11, Oct./Nov. 2008 535–555

DOI: 10.1177/0037549708096134



(a) Configuration #1 (b) Configuration #2

Figure 1. Example of when abstraction may decrease simulation performance

In practice, simulation-based analysis of large and complex systems is typically performed in phases, where each phase focuses on a specific aspect or scenario of a given system [1]. In other words, 90% of the time is spent in 10% of the model. Therefore, detailed data is typically required only from selected sub-models or specific scenarios. In addition, processing voluminous data from inconsequential parts of the model or scenarios merely exacerbates analysis [1]. In such focused studies, simulating the complete system using a high-fidelity, high-resolution model is unnecessary. Consequently, to improve the overall efficiency of simulation-based analysis, models at different levels of resolution are typically employed. Multi-resolution Modeling and Simulation (M&S) methods enable more optimal tradeoffs between observability, accuracy, fidelity and performance [1] for a given analysis. It is widely used in numerous fields to improve overall efficiency of the simulation-based analysis.

In our research, we have enabled multi-resolution simulations using hierarchical, component-based models and a novel methodology called Dynamic Component Substitution (DCS) [1]. In DCS, a set of components called a *module* is substituted with an equivalent component or vice versa to enable abstraction or refinement [1]. DCS transformations may be performed statically (i.e. prior to commencement of simulation) or dynamically (i.e. during simulation) to change the resolution of the model [1]. DCS can be used to achieve a wide range of model abstractions [1]. DCS transformations are governed by a DCS algebra that circumscribes changes to a model. DCS algebra also provides a mathematical framework for reasoning about changes induced by a sequence of transformations. A more detailed overview of DCS is presented in Section 3.

Typically, abstractions are used to improve simulation performance while trading-off observability and possibly some fidelity [1]. However, in many scenarios abstraction negatively impacts simulation performance and wall-clock time for simulation increases [1]. For example, consider the digital logic circuit shown in Figure 1(a). Figure 1(b) illustrates the scenario in which the 2 cascading AND-gates (namely B and C) constituting the *module* E have been abstracted into a single 3-input AND-gate *component* E. Each individual AND-gate has a delay of 5 units while the module consisting of 2 cascading AND-gates has a delay of $5+5 = 10$ units. Note that the functionality of the two configurations is equivalent [1, 6]. In concordance with conventional discrete-event simulation practices [6, 7], each gate processes the set of events presented at its inputs at a given simulation time and generates an output event with suitable timestamps.

Figure 1 also illustrates a typical sequence of events that are generated in the model. The events are indicated by $\vec{n}$, where n denotes the virtual timestamp on the events. As shown in Figure 1(a), the sequence of events in the model occur such that module D processes only a single event (at simulation time $\vec{10}$). On the other hand, as shown in Figure 1(b), when the AND-gates are abstracted, the abstract module (module E) introduces an additional output event (at simulation time 15) causing module D to process 2 events instead of 1. Note that this behavior conforms to standard hardware logic simulation procedures stipulated in the Language Reference Manual (LRM) for VHDL, one of the most widely-used hardware description languages [6, 7]. If the event granularity (wall-clock time taken to process an event) of module D is high, then the abstraction shown in Figure 1(b) actually deteriorates the overall performance of the simulation. Similar counter-intuitive cases have been observed in other large, commonly-used circuits such as 32-bit full adder, 32-bit multiplier and other Arithmetic and Logic Unit (ALU) circuits [1].

Identifying and avoiding performance-degrading abstractions in these larger models is a complicated and time-consuming task. Furthermore, in parallel simulations the impact of abstractions on parallelism (or concurrency) changes to model partitioning due to abstraction, communication costs and synchronization overheads also needs to be analyzed. For example, consider the Asynchronous Transfer Mode (ATM) network topology shown in Figure 2(a). The example network consists of two end clients interconnected using a pair of ATM switches. The ATM cloud (see Figure 2(a)) represents a higher level abstraction of the pair of the ATM switches. This is a typical, conceptual organization of ATM networks [1, 8, 9]. Corresponding parallel simulation-time layout consisting of logical processes (LPs) for the model on two workstations is shown in Figure 2(b) and Figure 2(c). Figure 2(b) illustrates the high-resolution version of the

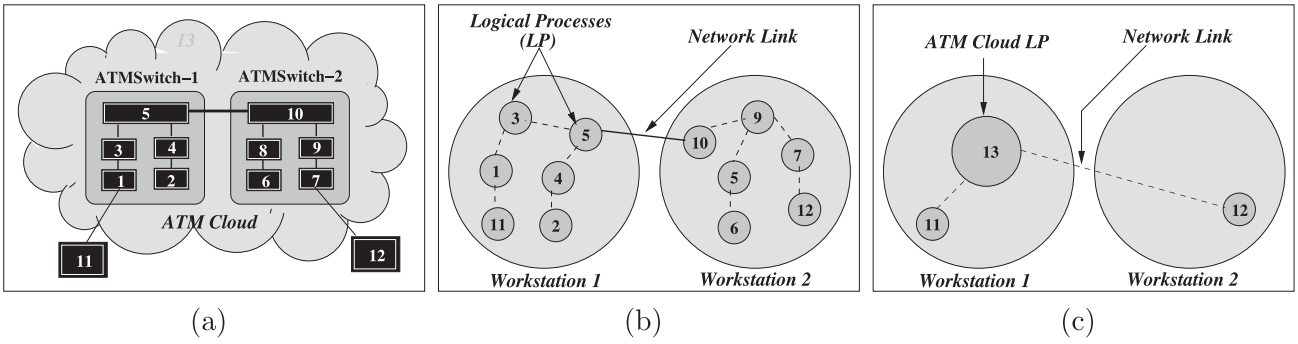


Figure 2. Example illustrating change in parallel simulation configuration due to DCS: (a) example ATM network; (b) high-resolution model; and (c) low-resolution model

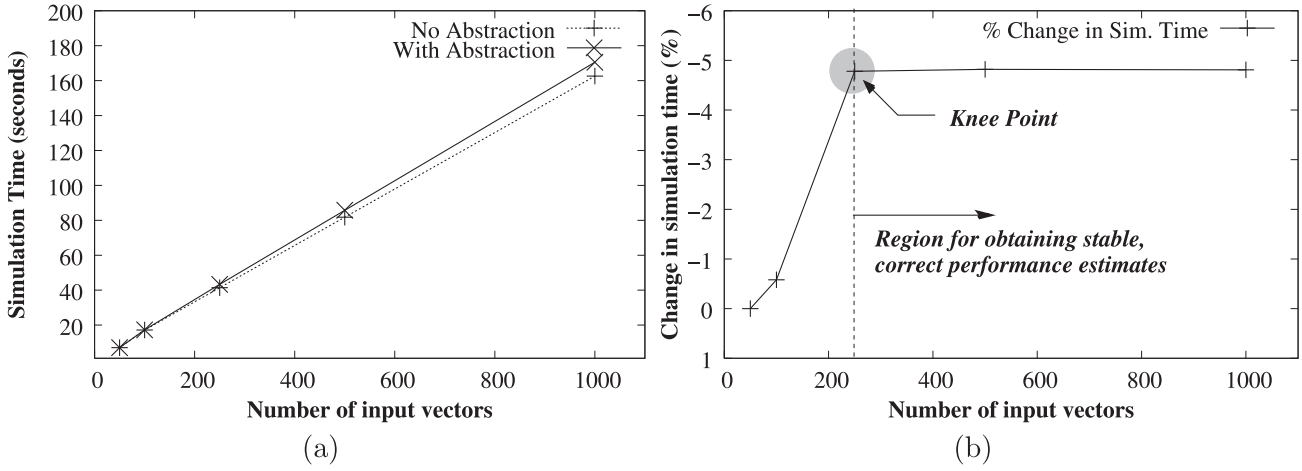


Figure 3. Observed change in simulation time due to abstractions: (a) wall clock time for simulation; and (b) percentage change in simulation time

network while Figure 2(c) presents an equivalent low-resolution configuration achieved via DCS. Although the lower resolution model has fewer components, the DCS transformation impacts the aforementioned parallel simulation parameters and causes degradation in performance. In other words, from a simulation performance perspective, it is beneficial to simulate the model in high resolution.

The complexity involved in identifying useful abstractions forces simulation practitioners to use 'dry runs' (or test simulations) to isolate effective abstractions. The dry runs are performed with selected resolution configurations using few input vectors. However, the primary problem with this approach is that the test simulations must be run for a sufficiently long time or using an adequate number of input vectors [1]. For example, consider the graphs shown in Figure 3. Figure 3(a) shows the wall clock time for simulating a model for a varying number of test vectors with and without a given abstraction. The percentage change in wall clock time is shown in Figure 3(b). As indicated by the graph in Figure 3(b), a stable performance observation can be made only after the 'knee point' when the model has been exercised with sufficient number of input vectors. If observations were made using fewer input vectors, there would be a significant skew in the observed data and the inferences drawn would be incorrect. On the other hand, identifying such knee points after which the data stabilizes may require numerous dry runs. Furthermore, comparisons must be performed using the average run times from several runs (typically 5–10) in order to provide sufficient confidence in the observations.

The additional steps involved in identifying performance-enhancing abstractions in the M&S cycle are shown in Figure 4(a). As shown in the figure, the modeler needs to perform numerous dry runs and comparisons to identify abstractions that improve performance. More-

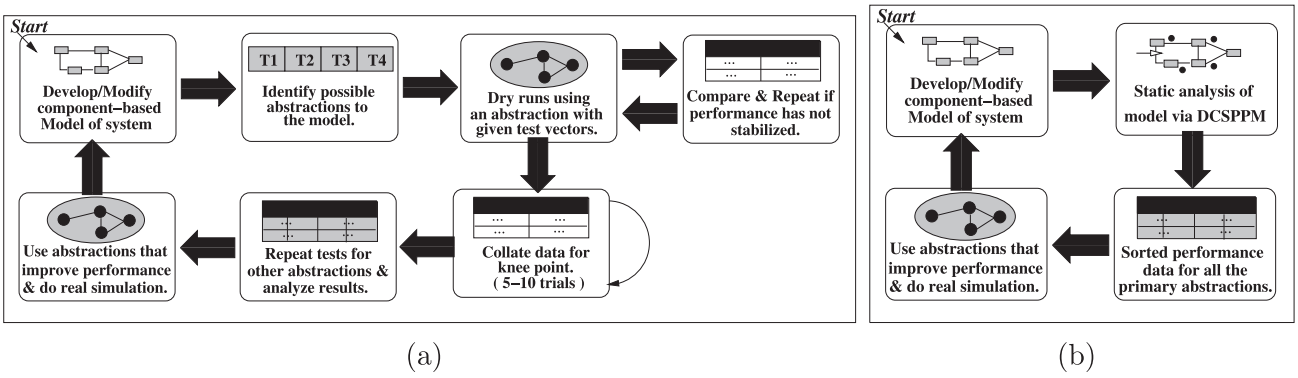


Figure 4. Overview of typical Modeling and Simulation (a) without DCSPPM (b) with DCSPPM

over, each time the model is changed, this task may need to be repeated. The aforementioned issues significantly hinder effective use of multi-resolution simulations. Consequently, effective approaches to rapidly identify effective abstractions and shrink the existing M&S cycle (see Figure 4(a)) are exigent [1]. Furthermore, in an existential discussion within the parallel and distributed simulation community [10], the viability of performance estimation techniques for parallel simulation protocols has been pointed out as being *critical* for the future success and general acceptance of parallel simulations [10, 11].

We have been investigating approaches to predict performance impacts of resolution changes and ease effective use of multi-resolution models, particularly in DCS-based simulations. Our endeavors have resulted in the development of a novel DCS Performance Prediction Methodology (DCSPPM). DCSPPM uses simulation platform-specific estimates of components and static analysis of a model to predict the performance impacts due to a given set of DCS transformations. Simulation platform estimates of a component are collated using one-time, granularity measurement simulations and cached for repeated use.

DCSPPM yields quantitative estimates that can be compared and used to identify performance-enhancing transformations. For example, given two transformations, say τ_1 and τ_2, DCSPPM yields estimates (in the form mean $\pm$ variance) e.g. $4.5 \pm 1\%$ and $-2.3 \pm 0.5\%$, respectively. Positive estimates indicate improvements in performance while negative values indicate degradation in performance. Using the estimates, a modeler can scientifically choose suitable model transformations to improve overall efficiency of simulation-based analysis. The significantly shortened M&S cycle when using DCSPPM is shown in Figure 4(b). As illustrated by the figure, DCSPPM directly uses the model and generates estimates for all primary abstractions. A modeler may also interactively determine performance impacts due to a combination of the primary abstractions. Once suitable abstractions are identified, they are applied to the model and simulation commences. The key point to note is that for a given transformation DCSPPM takes a significantly shorter time (orders of magnitude faster in most cases) when compared to a single analogous dry run, as shown in Figure 4(a). As illustrated by Figure 4, DCSPPM considerably reduces M&S cycles by easing identification of performance-enhancing abstractions in multi-resolution simulations.

This paper presents the issues involved in the design and implementation of DCSPPM along with an empirical evaluation. A brief description of some of the closely related research activities is presented in Section 2. Section 3 presents an overview of DCS. Our investigations have been conducted using a DCS-capable modeling and Time Warp synchronized parallel simulation environment referred to as Web-Enabled Simulation Environment (WESE). A brief overview of WESE and its DCS infrastructure is presented in Section 4. Section 5 presents a detailed description of DCSPPM along with the issues involved in implementing DCSPPM in WESE. This section also presents some of the statistical analysis performed to validate our implementation. The experiments conducted to evaluate the accuracy of the estimates generated by DCSPPM are discussed in Section 6. This section also presents experiments conducted to evaluate the sensitivity of DCSPPM to some of the external factors such as garbage collection rates, extraneous CPU load and network traffic. Finally, Section 7 concludes the paper summarizing the contributions from this work and providing pointers to future work.

2. Related Work

The techniques for performance estimation can be broadly classified into the four categories: (i) measurement; (ii) simulation; (iii) analytical modeling; and (iv) hybrid techniques [12]. Measurement is the most fundamental ap-

proach and is needed even in analytical or simulation-based techniques to calibrate the models. Simulation-based techniques involve constructing a model for the behavior of the system and driving it with an appropriate abstraction of the workload. Analytical modeling involves developing a mathematical model of the system behavior (at the desired level of detail) and analyzing it using mathematical tools.

Each approach has its own advantages and disadvantages and is more suitable for certain types of sub-systems than others. Consequently, to provide better system-level solutions, a mixture of the above-mentioned techniques are used. Such approaches, in which a mixture of performance estimation methods are used, are called hybrid techniques. DCSPPM is a hybrid technique that uses measurements and analytical technique for predicting performance impacts due to changes in resolution of a model used for parallel simulation. Accordingly, this section presents some of the closely related research activities in this area. Readers are referred to the literature [1] for a more comprehensive coverage of this topic.

Dickens et al. [13] present an analytical model of parallel discrete-event simulations for comparing the performance of YAWNS conservative synchronization protocol with Bounded Time Warp. Their analytical model is based on the assumption that the simulation is a heavily loaded queuing network where the probability of a server being idle is almost zero. They develop and validate analytical methods for computing approximated performance measures as a function of the degree of optimism allowed, cost of state-saving, rollback overheads and barrier synchronization and workload characteristics. Our research, on the other hand, deals with predicting performance impacts due to resolution changes rather than considering the simulation as whole.

Balakrishnan et al. [14] present a simulation-based Performance and Scalability Analysis Framework (PSAF) for performance prediction of any discrete-event simulator. Using PSAF, synthetic models can be easily developed for different simulators. The synthetic models are designed and fine-tuned to exercise the various aspects of the simulator. The models are simulated with various simulators and their performance is compared. The objective of PSAF is to compare different parallel simulators while DCSPPM is designed to compare different resolution configurations of a given model on a given parallel simulation environment.

Ferscha and Johnson [11] present N-MAP tool set, a performance prediction testbed for Time Warp simulations. The objective is to support performance engineering endeavors from an early design phase of Time Warp protocols in order to avoid late or costly re-engineering. Using the N-MAP toolset, a programmer starts with a rough description of the algorithmic structure in the form of a skeletal code. The skeleton is refined in an iterative manner by providing additional details about the program and analyzing the performance impacts at each incremental step. The performance of the program is estimated using simulations. The idea is to make optimal decisions at each incremental step so that the final parallel program runs with an optimal configuration. The N-MAP tool set is designed to identify optimal parameters (e.g. event structure, average LVT progression, commitment rate and state saving) for a Time Warp simulator.

An analogous research reported by Liu et al. [15] uses an hybrid approach for performance estimation of a conservatively synchronized parallel simulator called DArtmouth implementation of the Scalable Simulation Framework (DaSSF). In their work, they empirically estimate the various overheads in their parallel simulator such as context switching, dynamic object costs, procedure call overheads, dynamic channel overheads, cost of process orientation, event list management costs and a limited estimate of synchronization costs. The empirical estimates are then suitably combined based on intimate knowledge of the simulator and the model to predict simulation performance. The experiments conducted in context of their studies (i.e. models of network simulations developed using the SSF API) show that the error in their approach is less than 10%. Unlike N-MAP and DaSSF, the objective of DCSPPM is to identify optimal resolution configurations from a subset of feasible transformations. Contrary to N-MAP and DaSSF, DCSPPM uses a combination of measurement and automatic static analysis of a model. On the other hand, similar to N-MAP, DCSPPM is also built on the assumption that the parallel simulator uses the Time Warp protocol for synchronization.

Gupta et al. [16] present a discrete-state continuous-time Markov chain model for performance analysis of Time Warp simulations. They make several assumptions about a number of simulation characteristics and have validated their model using a Time Warp testbed executing on a shared-memory multiprocessor workstation. Tay et al. [17] present an analytical model to evaluate the performance of Time Warp simulation with cascading rollbacks. They propose performance metrics for various simulation characteristics such as rollback probability, rollback distance, elapsed time, cascading rollbacks and Time Warp efficiency including throttled Time Warp. These research activities deal with performance evaluation of Time Warp simulations. In contrast, the proposed research deals with predicting the performance impact of resolution changes in Time Warp synchronized, multi-resolution simulations.

Perumalla et al. [18] recently presented a virtualization system that is designed to help simulation-based performance prediction of parallel/distributed discrete-event simulations performed using supercomputing platforms. It is intended to be useful in experimenting with and understanding the effects of execution parameters, such as different load balancing schemes and mixtures of model fidelity [18]. Our research differs from theirs in that we do not use virtualization but operate directly on the final model. Furthermore, DCSPPM uses hybrid techniques rather than a simulation-based approach for performance

prediction. A number of other similar research activities related to performance estimation are available in the literature [1].

3. Dynamic Component Substitution (DCS)

Dynamic Component Substitution (DCS) is a generic methodology for changing the resolution (or level of abstraction) of any hierarchical, component-based model [19, 20]. In a component-based model, a system is represented as a set of interconnected components [19, 20]. A component is a well-defined entity which is viewed as a 'black box', i.e. only its interface and functionality are of interest and not its implementation. However, during simulation, each component is associated with a logical process (LP) or a simulation object (a specific, well-defined software unit) that implements its behavior and functionality. A component could in turn be specified using a set of sub-components. A set of interconnected components with a well-defined interface is called a *module*. A module can be viewed as a logical component. Modules are conceptual, convenient abstractions that enable development of hierarchical models and ease modeling of large systems [19].

In DCS, changes to the resolution of a model are achieved by substituting a module (i.e. a set of components) with a functionally equivalent module or vice versa. The equivalent module is chosen such that the overall characteristics of the model are not altered and deviations in the simulation results due to DCS are within acceptable limits. The equivalent module may contain one or more components. In practice, the equivalent module typically contains a single component. In such cases, the module is simply called an equivalent component. An equivalent module or component must satisfy the following criteria.

1. Interface equivalence: Equivalent modules must have an interface that is identical. The interface specification also includes the type of events that can be processed or generated by the LP associated with the component. The interface equivalence property ensure that DCS transformations of a module are completely transparent to any other interconnected component.

2. Functional equivalence: The equivalent modules must have similar functionality, as defined by the modeler. In other words, the behavior and functionality of a given pair of equivalent modules need not be identical but within acceptable limits. This flexibility enables the application of different forms of abstractions to the model. This criteria is a weaker requirement then interface equivalence.

Substitution of components may be done *statically* or *dynamically*. Static substitutions occur during model development or prior to commencement of simulation. Static component substitution is widely used in different flavors, to address capacity and performance issues of large-scale simulations [21, 22]. The primary drawback of the static methods is that the tradeoff between fidelity, resolution and simulation overheads cannot be altered during simulation. On the other hand, substituting components during simulation enables a dynamic tradeoff between the modeling and simulation-related parameters. The dynamic flavor of DCS can be further classified into the *Proactive* and *Reactive* DCS transformations. Approaches in which DCS transformations are scheduled to occur in the future (with respect to current simulation time) are classified as proactive strategies. In reactive strategies, however, DCS transformations are scheduled at the current simulation time or in the past. Proactive transformations are used to drive the model to known optimal configurations while reactive transformations are used to recover from recent incorrect proactive transformations.

DCS can be used to enable dynamic (i.e. during simulation) tradeoffs between fidelity and simulation performance by changing model resolution during simulation. DCS has been applied to accelerate simulations of a diverse set of systems such as: estimating power dissipation of VLSI circuits [23]; analyzing rare phenomena or rare events in detailed Asynchronous Transfer Mode (ATM) network simulations [24]; simulation of spatially explicit models of Lyme disease epidemiology [25]; and Mobile Ad hoc Networks (MANETs) [1]. Each of these applications involved different forms of abstractions [1]. Detailed descriptions of each application and use are available in the literature [1, 8, 23, 24, 25]. However, analogous to any multi-resolution approach, care must be taken when applying DCS to avoid several modeling and simulation pitfalls such as: temporal inconsistencies; ghosting of attributes; high transition latency; thrashing; and degradation in fidelity [1, 26].

One of the most attractive and important aspects of DCS is that algebra has been developed for reasoning about the changes induced in model by DCS transformations [27]. The algebra has its foundations in discrete mathematics and set theory. The algebra clearly defines and delineates the various transformations that can be induced in the model. Based on the axioms underlying the algebra, it has been proven that a sequence of DCS transformations satisfy the following properties.

1. Closure: The set of all possible transformations is finite.

2. Associativity: Given any three DCS transformations τ_1, τ_2 and τ_3, $(\tau_1 \circ \tau_2) \circ \tau_3 = \tau_1 \circ (\tau_2 \circ \tau_3)$, where $\circ$ is a binary operator that represents the application of two given non-commutative transformations to a model.

3. Identity: DCS algebra defines an identity transformation τ_ϕ such that $(\tau_i \circ \tau_\phi) = (\tau_\phi \circ \tau_i) = \tau_i$.

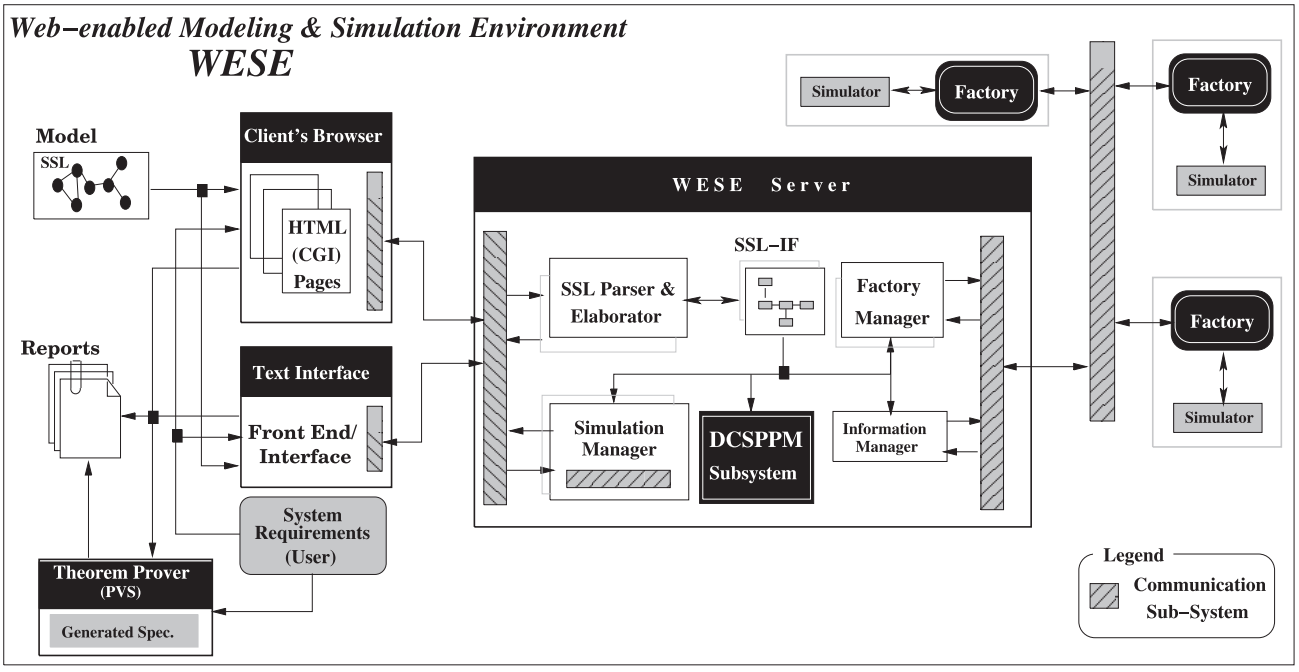


Figure 5. Overview of WESE

4. Inverse Property: Given a DCS transformation τ_i, there exists an inverse transformation τ_i^{-1} such that $(\tau_i \circ \tau_i^{-1}) = (\tau_i^{-1} \circ \tau_i) = \tau_\phi$.

Since DCS algebra satisfies the aforementioned properties, it constitutes a 'group' in discrete mathematics [28]. These properties also play an important role in efficient implementation of DCSPPM. For instance, associativity implies that transformations can be carried out in different orders without violating their validity. Similarly, from the identity property, if follows that NULL (τ_ϕ) transformations can be eliminated from estimation because they do not change the model. These aspects make DCS a rigorous and efficient approach for performing multi-resolution simulations. Further details on DCS and DCS algebra along with mathematical proofs of properties are available in the literature [1, 27, 29, 30].

4. Web-enabled Modeling and Simulation Environment (WESE)

This section presents a brief description of WESE [19, 27, 29], the web-enabled, component-based modeling and simulation environment which has been used to conduct this research. An architectural overview of WESE is shown in Figure 5. As illustrated in the figure, WESE provides both a HTML interface and a text-based frontend for user interaction. The primary input to WESE is the model of the system described using the System Specification Language (SSL), which we briefly discussed using an example. Figure 6 presents the SSL description of a 3-input AND-gate module [1].

As shown in Figure 6, the specification of a model or a SSL design file consists of a set of interconnected *modules*. Each module consists of three main sections, as follows.

1. Component Definitions: Defines the set of components used in a module and the source of LPs for each component. The number of input and output ports for components is also defined.
2. Component Instantiations: Specifies the instances of components, defined in the component definitions section, that will be used to describe the model. Using high level programming languages as an analogy, component definitions are similar to type definitions while component instantiations are analogous to variables of a given type.
3. Netlists: This section specifies the interconnection between ports of components. Interconnection specifies the logical flow of events in the model.

As shown in Figure 6, SSL permits an equivalent component (see Section 3) to be associated with each module. DCS is performed by replacing the module with its equivalent component or vice versa. SSL also allows an optional *label* (example AndG3M in Figure 6) to be associated with each module. The label can be used as a

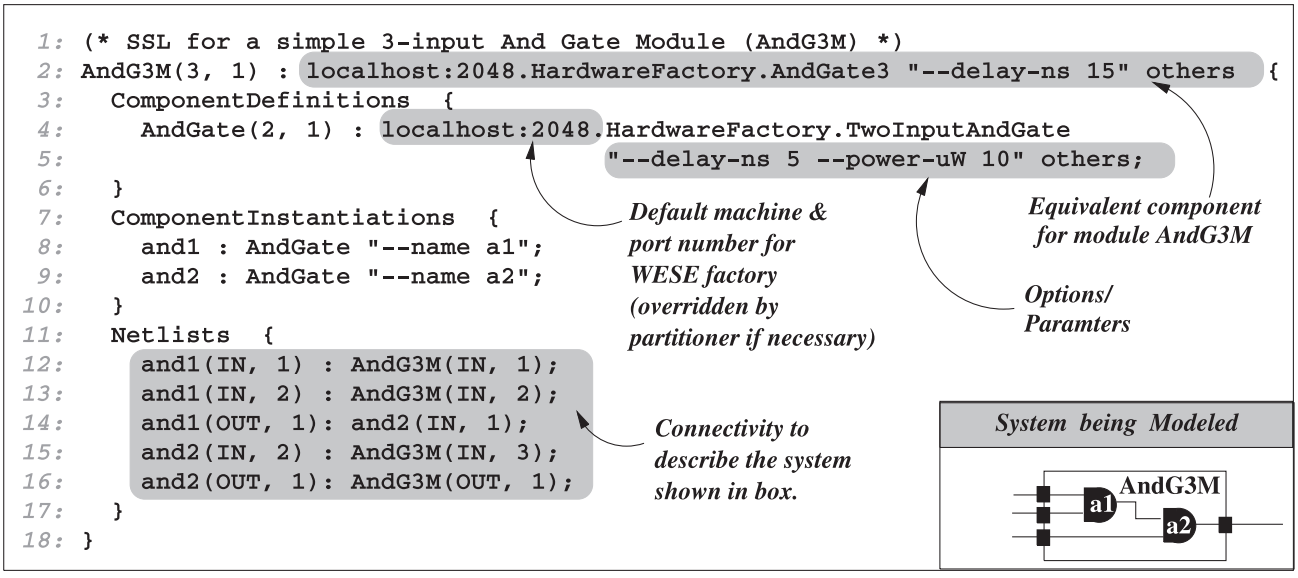


Figure 6. Example SSL source file for a three-input AND-gate module (AndG3M)

component definition in subsequent module specifications to nest one module within another. This technique can be employed to reuse module descriptions and develop hierarchical specifications.

As shown in Figure 5, the input SSL source is parsed into an object-oriented (OO) in-memory *intermediate form* (SSL-IF). SSL-IF is the primary data structure used by DCSPPM for static analysis of the model. Figure 7 illustrates the core classes constituting SSL-IF. As illustrated by the figure, object-oriented features of C++ language have been employed in the design of SSL-IF. The design objective of SSL-IF is to provide efficient access to related data from different parts of a model. All the SSL-IF classes have been derived from SSL_Base class. This aspect enables construction of SSL-IF through pointer composition. Operations performed using SSL-IF are implemented using polymorphic virtual methods in each of the classes. The _dcsppm method (see Figure 7) performs all the phases of DCSPPM. When this method is invoked on a SSL_DesignFile instance, it invokes appropriate methods on each module in the design file. Intermediate results are selectively stored in member objects (such as cachedBR in Figure 7) to reduce overall analysis time. A more detailed description of the data structure, its design and implementation is available in the literature [1].

The core sub-system of WESE is the server (Figure 5). The WESE server performs the task of collaborating with the distributed factories and coordinating the simulations. The *factory manager* performs the tasks of interacting with the distributed WESE factories using a predefined protocol. A WESE factory can be viewed as a web-based repository of components with added capability to simulate them. Parallelism occurs at the factory level i.e. each factory is a parallel, asynchronous simulation entity [19]. Parallel simulations are performed by utilizing components (or simulation objects) from different factories. A WESE factory is built from sub-factories and *stubs*. Stubs contain attributes of a component such as interface description, cost and formal specifications.

The *simulation sub-system* of a WESE factory has been developed using the WARPED simulation kernel. WARPED is an Application Program Interface (API) for a general purpose discrete-event simulation kernel with different implementations [31]. WESE utilizes the Time Warp [31] based simulation kernel of WARPED. A Time Warp synchronized simulation is organized as a set of asynchronous LPs that represent the different physical processes being modeled. The LPs exchange event information by exchanging *virtual time*-stamped event messages. Each LP processes its events by incrementing a local virtual time (LVT), changing its state and generating new events.

Although each LP processes local events in their correct time-stamp order, events are not globally ordered. Causality violations are detected when an event with time-stamps lower than the current LVT (a *straggler*) is received. On receiving a straggler event, a rollback mechanism is invoked to recover from the causality error. The rollback process recovers the state of the LP prior to the causal violation, canceling the erroneous output events generated and re-processing the events in their correct causal order. Each LP maintains a queue of state transitions along with lists of input and output events cor-

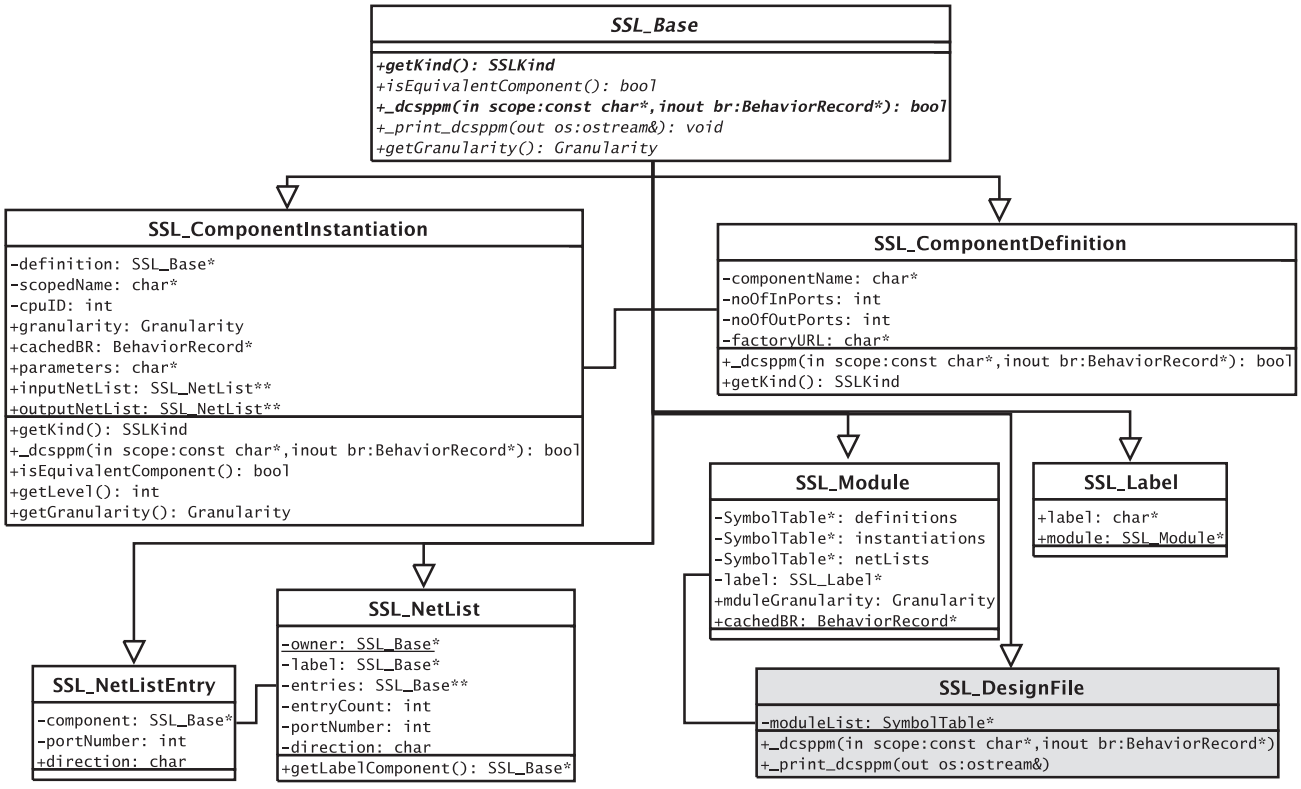


Figure 7. Overview of classes constituting SSL-IF

responding to each state to enable the recovery process. A periodic garbage collection technique based on Global Virtual Time (GVT) is used to prune the queues by discarding history items that are no longer needed. The distributed simulation is deemed to have terminated when all the events in the system have been processed in their correct causal order. A more detailed description of WARPED, Time Warp and WESE are available in the literature [31, 32].

The modeling and simulation sub-systems of WESE provide the infrastructure for enabling both proactive and reactive DCS. In WESE, an event-driven mechanism has been employed to sequence the various phases involved in DCS [33]. WESE provides API calls to schedule proactive DCS transformations which will occur in the future. Reactive DCS transformations are achieved by artificially rolling-back the simulation to an earlier simulation time (i.e. earlier than the time when the transformation needs to occur) and then performing the DCS transformation. The overheads of rollbacks and causal consistency maintenance are transparent to the application. In order to enable reactive DCS, the state of the simulation at the desired time needs to be available. To ensure that the states are available, WESE provides suitable API calls that can be used to delay garbage collection in the simulations. Care must be taken to ensure that an optimal value is specified so that the memory usage of the simulation does not significantly increase.

In addition, WESE provides an API and infrastructure for mapping states of components during DCS. WESE also includes a lightweight Error Propagation Library (EPL) that provides a set of statistical functions that can be used to track and propagate errors in simulation results that may arise due to DCS transformations. EPL provides a number of statistical functions that can be used to determine the uncertainty or errors. Operator overloading has been used to define suitable mathematical operators to automatically propagate the uncertainties through the model. Experiments have indicated that EPL introduces minimal (less than 1) performance penalties. Details regarding experimental evaluation of EPL are available in the literature [1].

5. DCS Performance Prediction Methodology (DCSPPM)

DCSPPM aims to provide a quantitative measure of the *change* in simulation time due to a given set of DCS transformations for a given partition of the model. An overview

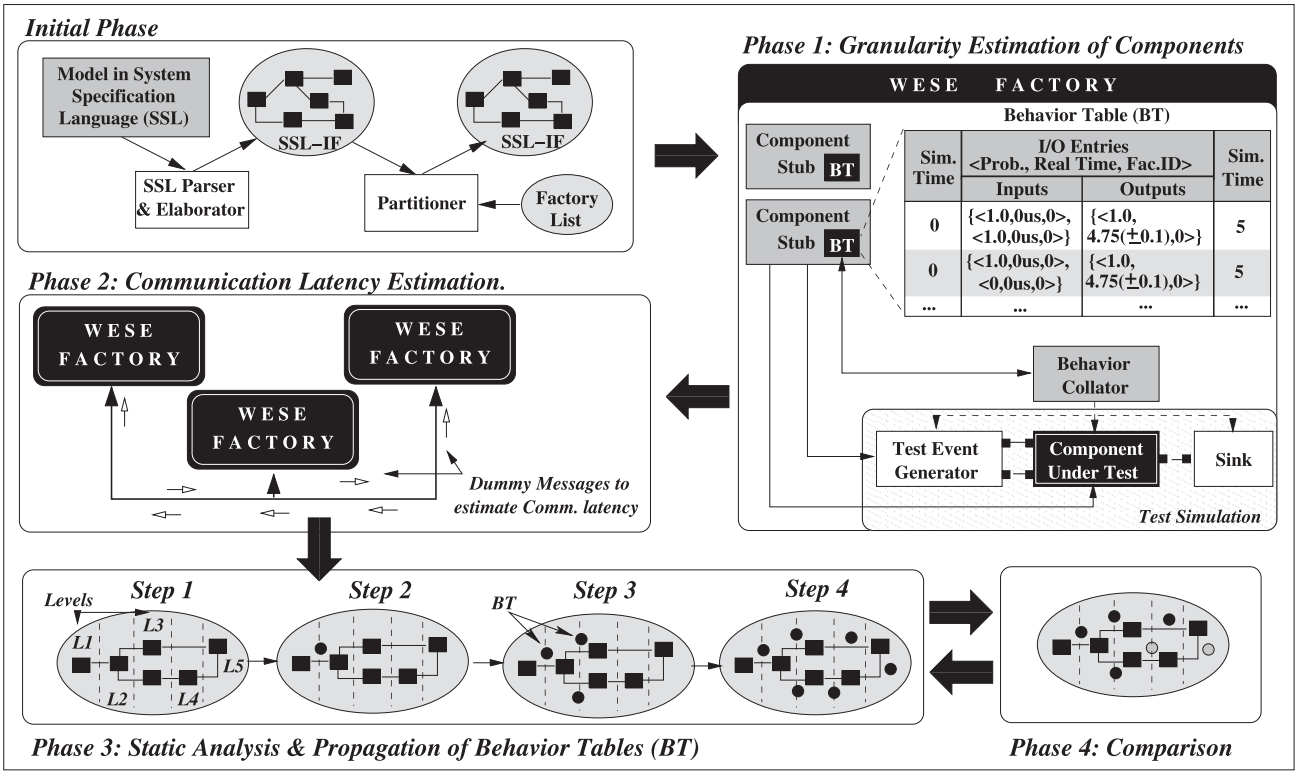


Figure 8. Overview of DCSPPM

of the DCSPPM is shown in Figure 8. As shown in the figure, component-based models are parsed into an object-oriented, in-memory intermediate format called SSL-IF. The SSL-IF is partitioned by logically assigning components to a given set of WESE factories. Currently, the partitioning is random and assigns equal number of components to each factory used for simulation. The partitioned SSL-IF is utilized by the DCSPPM module for further processing. As shown in Figure 8, DCSPPM proceeds in four distinct but overlapping phases. A discussion of the four phases is presented in the following subsections.

5.1 Phase 1: Collating Behavior Tables for Components

The first phase of DCSPPM (see Figure 8) involves collating the data for generating the Behavior Table (BT) for each atomic component. The BT of a component essentially consists of several rows indicating the input-output characteristics of the component. Each row specifies the probability of output vectors (or events) being generated by the component at each output port for a given combination of input vectors (or events) at its input ports. Components that do not have any inputs are treated as a special case and are described using a reserved NULL input vector. Similarly, components that do not have any outputs have a NULL output vector. The set of I/O vectors are implicitly ordered to reflect the order of ports in a component to minimize the size of BTs. The I/O vectors are also grouped based on their virtual time-stamps to ease analysis in Phase 3. As shown in Figure 8, the I/O vectors at each port are represented using a 3-tuple consisting of <I/O Probability, Real Time, Factory-ID>. The three components of an I/O vector are discussed in the following subsections.

5.1.1 I/O Probability

The probability value associated with an I/O vector essentially indicates the probability with which an event occurs at a port. Probability of 0 indicates absence of an event. In WESE, the I/O probability value may be empirically determined using test simulations (see Figure 8). Such test simulations are effective for components with few input and output ports. All possible combinations of input vectors are fed to the component and the resulting output is analyzed to determine I/O probability. This approach has been used for collating BTs for the components that have 2–5 input ports. For example, logic components used for developing the circuits used in this study utilize this approach.

However, such an approach does not scale for large components or components whose number of I/O ports is not fixed at model development time. In such cases, it is the responsibility of the modeler to use suitable API calls to appropriately specify the I/O probabilities. Furthermore, to address scalability issues, the BT entries are computed on-the-fly as needed. This feature is used to describe the behavior of Asynchronous Transfer Mode (ATM) network switch (ATMSwitch) component used in this study. In this case, the stub (see Section 4) associated with the ATMSwitch, provides BT entries on demand, once the actual model to be analyzed is known. Furthermore, it computes output probabilities based on model specific knowledge that if an input event arrives at one of its inputs, the switch generates an output with equal probability at one of its output ports. In other words, for a k-port switch, an input vector with probability of p will result in an output at each output port with a probability of p/k.

5.1.2 Real Time

This field indicates the real time (or wall clock time) at which the I/O events occur. Alternatively, this value indicates the time taken to process the given input vector and generate outputs. For example, the first row of the BT shown in Figure 8 indicates that if an input vector is presented to the component at real time 0, it generates output at 4.75 ± 0.1 μs. Granularity also includes the simulation kernel overheads for processing the event such as event scheduling costs, time spent for state saving and garbage collection overheads. However, it does not include communication overheads or any synchronization overheads.

It is the responsibility of the modeler to specify the set of 'typical' events to be used for granularity estimation via suitable API calls. Typical events are those events that would be most commonly processed by the component and represent its average or characteristic behavior. For example, the ATMSwitch processes a diverse set of events. However, the most common events pertain to network traffic or cells. Accordingly, cell events are used to estimate the Typical Event Granularity (TEG) of the ATMSwitch. Once the typical events are specified, the rest of the process of estimating TEG is fully automated in WESE.

TEG is assumed to follow a Normal distribution in concordance with statistical theories [12, 34]. Weighted averages are used to determine TEG for components that have more than one significantly different TEG. For example, consider the code snippet shown in Figure 9. Assume that the cases labeled KIND1 and KIND2 have individual event granularities of $N(\mu_1, \sigma_1^2)$ and $N(\mu_2, \sigma_2^2)$ respectively, where $N(\mu, \sigma^2)$ represents a Normal distribution with mean μ and variance σ^2. Given their weighted probability of the occurrence to be p and $q = (1 - p)$ respectively, the overall TEG is calculated as $\left[p \times N(\mu_1, \sigma_1^2)\right] + \left[q \times N(\mu_2, \sigma_2^2)\right] = N(\left[p \times \mu_1\right] + \left[q \times \mu_2\right], \left[p^2 \times \sigma_1^2\right] + \left[q^2 \times \sigma_2^2\right])$. This technique can be also extended to components with three or more distinct event granularities. Proof of validity of this operation is available in the literature [34].

Table 1. Results from statistical tests for NotGate component: left-stem plot (note that the decimal point is 2 digit(s) to the left of the |)

530	5
532	2679
534	031
536	680
538	122233456
540	001278968
542	024
544	0133114
546	124550
548	14
550	898

Table 2. Results from statistical tests for NotGate component: Shapiro–Wilk's test

data	x
W	0.9792
p-value	0.5176
W	$>$0.5
p-value	$>$0.05
Cannot reject null hypothesis H_0: Population is normal	

Given the typical events, WESE provides the infrastructure for automatically analyzing the granularity data, extracting distinct event granularities and performing the averaging operations. However, it is suggested that the assumption of normality be verified [12, 34] using a combination of several statistical techniques such as box plots, histograms, Q-Q plots, stem and leaf plots and Shapiro–Wilk's test.

Figure 10 illustrates these plots for a logic gate (NotGate) component obtained using an open-source statistical package called $\mathcal{R}$ [35]. As shown in Figure 10(a), the box plot reflects a standard normal distribution. The histogram shown in Table 1 and the Leaf–Stem plot shown in Figure 10(b), provide strong evidence of normality. The observed data points are sufficiently close to the theoretical value as shown in the Q-Q plot in Figure 10(c). The high p-value (0.5176) and large W value (0.9792) from Shapiro–Wilk's test (Table 2) clearly demonstrate that the granularity values follow a normal distribution. A more detailed discussion of the statistical tests, their inferences and results for each one of the components used for empirical evaluation in this study is available in the literature [1].

The primary motivation for normality verification is to ensure that the modeler has specified a sufficient number

```
void
SampleComponent:::executeProcess()  {
    do  {
        SampleEvent *e = (SampleEvent *) getEvent();
        switch (e–>getKind())  {
        case SampleEvent::KIND1:
            doProcedure1();
            // Granularity = 4.875 microsecs
            break;
        case SampleEvent::KIND2:
           doProcedure2();
           // Granularity = 10.732 microsecs
           break;
        }
    } while (haveMoreEvents() == true);
}
```

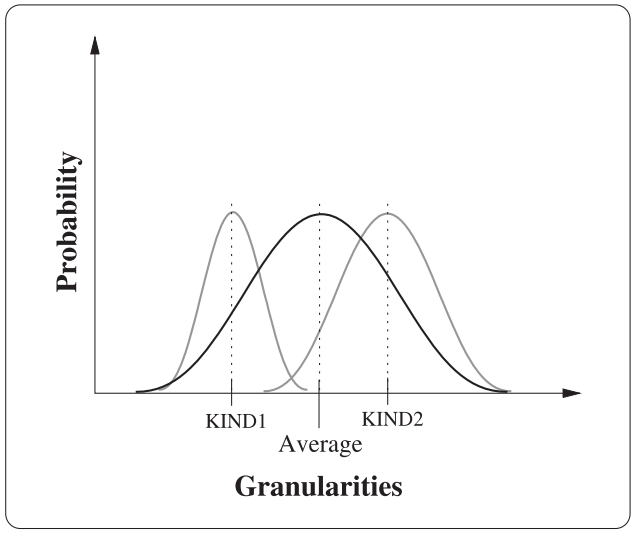


Figure 9. Example of averaging multiple distinct granularities

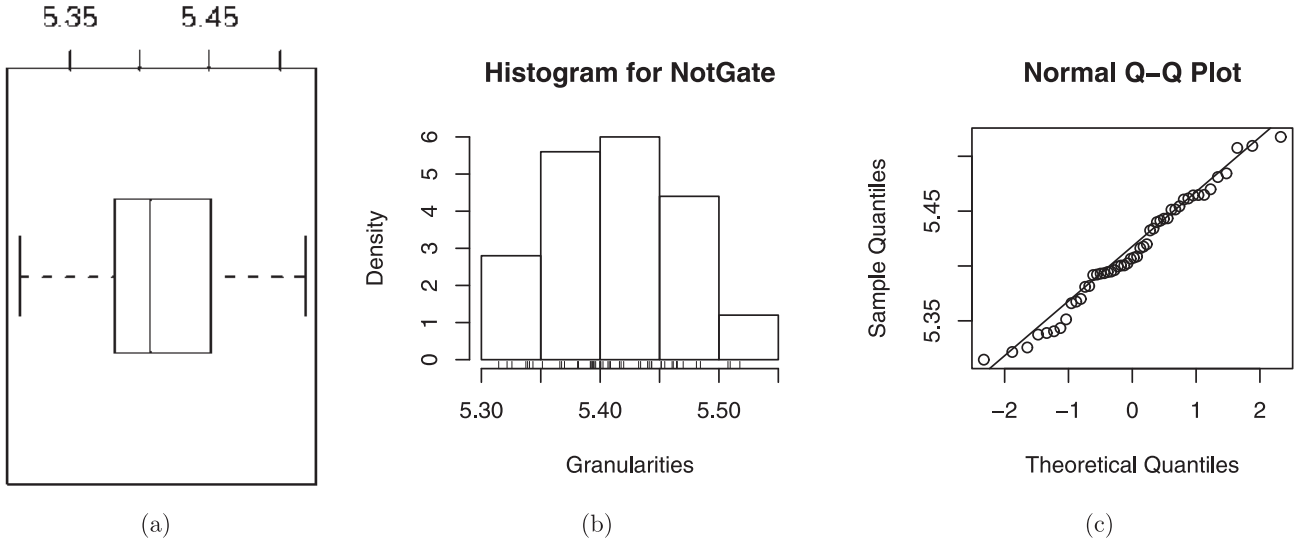


Figure 10. Results from statistical tests for NotGate component: (a) box plot; (b) histogram; and (c) Q-Q plot

of samples to be collated by WESE to yield a good representative statistic. Although a modeler can safely specify 100 samples (which is statistical infinity in most cases [34]) for each typical event, this may unnecessarily increase estimation times. For example, average granularities from 40 test events were more than sufficient to compute the TEG for digital logic components used in this study. On the other hand, a minimum of 65 test events were needed to obtain good normality for the ATMSwitch component. The smaller the value, the faster is the estimation. However, care must be taken to ensure the values are not too small to minimize statistical errors.

5.1.3 Factory-ID

The third value in the tuple indicates the logical WESE factory ID to which the component has been assigned by the partitioner. The ID value is used (in Phase 3) to detect and track interactions between components on different factories which require communication over the

network. Tracking such network centric communication points serves the following two purposes in DCSPPM.

1. It enables appropriate inclusion of communication latencies that impact the overall TEG of a model.
2. It is used to identify points where potential for rollbacks exist in order to account for synchronization overheads in Time Warp simulations. More specifically, in WESE, rollbacks occur only when events are received over the network. A more detailed discussion of the heuristics used to track rollbacks is discussed in Section 5.3.

5.2 Phase 2: Estimating Network Latencies

Estimation of communication latencies is performed using a pair of WESE factories only when more than one WESE factory is used for simulation. One WESE factory acts as a server while the other acts as a client. Communication latency is estimated by exchanging a large number of messages between the two factories and measuring the Round Trip Time (RTT) for the messages. A number of the RTT measurements are averaged to obtain the mean latency and variance. Similar to component granularities, the average latency value is assumed to follow a normal distribution. The estimation process is suitably coordinated by the DCSPPM Module (see Figures 5 and 8). The estimated values are stored in the DCSPPM Module and reused as necessary during Phase 3.

5.3 Phase 3: Estimating Granularity of a Module

The overall TEG of a module is estimated in a recursive, top-down manner using the TEG of each component (estimated in Phase 1) and sub-module constituting the module. The estimation is performed by propagating the BTs of components from inputs to outputs of the model. As the BTs are propagated they are suitably transformed to include the behavior and characteristics of the components constituting a module. SSL-IF, the in-memory intermediate form, is the core datastructure used for this phase. Results from computations performed during this phase are stored in SSL-IF. As illustrated in Figure 8, this phase proceeds in the following steps.

Step 1: Levelization: The objective of levelization is to capture the natural flow of events from inputs to outputs of a module. Accordingly, in this step, each component or sub-module is assigned a level number such that other components that generate inputs to a given component are at a lower level. Cycles (output of a higher level is fed as input to a lower level component) in a module are identified and arbitrarily broken. Currently, DCSPPM does not effectively account for cycles in a model. Levelization results in updates and minor reorganization of SSL-IF, the in-memory intermediate representation (see Section 4) of the module.

Step 2: Propagation of BTs: In this step, the BTs of components are propagated from the lowest level (input) to the highest level (output) of a module. Every component in a level utilizes the set of input BT records (IBTRs), its own BT and generates output BT records (OBTRs) at its output ports. For each IBTR, the OBTRs are computed in the following manner.

1. First, the probabilities in a IBTR are used to detect the presence or absence of inputs at each port which yields an input pattern. The components' BT entry corresponding to the input pattern is obtained from the appropriate WESE factory. For example, consider a scenario in which a input BT entry $\{< 0.5, 10 \pm 1\mu s, 0 >, < 0.75, 12 \pm 1\mu s, 1 >\}$ at simulation time 35 is presented to a component whose BT is shown in Figure 8. In this case, the component has inputs on both ports and uses the corresponding entry from the BT. In this case, the first row is obtained from the WESE factory.
2. The output simulation time is computed by adding the IBTR simulation time to the value in the component's BT entry. In the above example, the output simulation time of the BR would be $35 + 5 = 40$.
3. Next, the maximum probability value from the IBTR is multiplied by the probability values in the component's BR entry to determine probability of OBTR. In the above example, the output probability would be: $1.0 \times \max(0.5, 0.75) = 0.75$.
4. If vectors arrive from different factories (identified using factory-id in IBTR entries) the communication latency between the pair of factories is added to the real-time value of the corresponding input vector. For example, assume that the component being analyzed is assigned to factory 1. Given the earlier input vectors, the first vector ($< 0.5, 10 \pm 1\mu s, 0 >$) is from factory 0. In this case, the communication latency, say $40 \pm 7\mu s$, is added to the real time (resulting in $< 0.5, 50 \pm 8\mu s, 0 >$). Note that the real-time operations use statistical arithmetic based on the fact that these values represent normal distributions with a given mean and variance [34].
5. The real time at which the outputs are generated is determined by adding the maximum real-time value of the IBTR vectors to each

OBTR vector. Based on the earlier example, the real time for the output event would be: $4.75 \pm 0.1 + \max(10 \pm 1, 12 \pm 1) = 16.75 \pm 1.1$.

6. Next, the following two heuristics are used to account for synchronization overheads in the simulation. Since the simulations are based on Time Warp, the heuristics generate a rollback factor. The heuristics are based on the fact that, in WESE, rollbacks occur only when inputs are received from multiple factories. Rollbacks require reprocessing of events which requires additional wall clock time. Accordingly, the rollback factor is used to suitably scale the overall real-time value of the output vectors. In addition, the rollback factors are propagated through the model to account for cascading rollbacks. Cascading rollbacks occur even although all events are received from the same factory, as some components in the input chain rollback.
7. *Heuristic 1*: This heuristic is based on the fact that if inputs at the same simulation time arrive at different real times from different factories, there is a probability of rollback. Accordingly, the real-time values of various input vectors from different factories are statistically compared and, for each pair that is different, the rollback factor is increased by one. The real-time values for each output vector are then suitably scaled using the rollback factor value for the component, i.e. realTime = realTime + (realTime × rollback_factor).
8. *Heuristic 2*: This heuristic uses the fact that if inputs at earlier simulation times arrive at a later real time when compared to one another, then rollbacks occur. For example, if inputs at simulation time 30 arrive at real time $25 \pm 1\mu$s while inputs at simulation time 40 arrive at real time $10 \pm 1\mu$s, then a rollback will most likely occur. The real-time values of successive input behavior record entries are statistically compared to determine rollback probability. Similar to the previous heuristic, the real-time values for each output vector are suitably scaled to reflect synchronization overheads.
9. For models with cycles, a two-stage approach is used for estimation. During the first stage, the available behavior records are used for behavior propagation. If no inputs are available, then the cycle is broken by arbitrarily choosing the NULL BR of a component and utilizing it to generate a temporary output behavior for a component. The temporary output behavior is propagated along the cycle during the first phase. In the second stage, the BTs from the first phase are further refined and updated to fully estimate the interaction and behavior between the components in the cycle.

Step 3: Overall Granularity: In order to determine the granularity of a module, first the components in a module are grouped based on the factory in which they reside. Next, the granularity contributed to each factory by the module is determined by adding the granularity of components in each group.

Estimation of granularity of components proceeds in a recursive, top-down manner until the top-level module has been analyzed. At the end of analysis, the overall granularity contribution to each factory involved in the simulation is collated in the SSL-IF node corresponding to the top-level module. Since each factory simulates in parallel, the maximum of these values is used as an estimation of the overall typical event granularity (TEG) of the model as a whole. For example, the result of this phase would be a TEG in the form $4567 \pm 38\mu$s. This TEG is retained as the reference for further comparisons in Phase 4.

5.4 Phase 4: Estimating Performance Changes due to DCS

In this phase, parts of the model that undergo DCS transformations are located and Phase 3 is repeated starting with that part of the model. However, rather than using the BT for the module, the BT for the corresponding equivalent component is used. This results in a TEG value that indicates the overall estimated granularity with the new component in place. For example, say the result is $4238 \pm 42\mu$s. The new TEG is compared with the reference TEG from Phase 3 to determine the change in performance due to a DCS transformation. Using the earlier example, the difference in performance would be $(4567 \pm 38 - 4238 \pm 42)/4567 \pm 38 = 7.2 \pm 1.9\%$.

5.5 Assumptions underlying DCSPPM

DCSPPM is a static performance prediction methodology. Several assumptions regarding the dynamic characteristics of the model and the simulation platform have been made during its design and implementation. The primary motivation for the assumptions is three-fold, namely: (i) to ensure the task at hand is tractable; (ii) to ease implementation of DCSPPM; and (iii) to optimize and reduce time for estimation. These assumption play an important role in the applicability and validity of the estimates generated by DCSPPM. The assumptions underlying DCSPPM are as follows.

1. One of the important requirements is that the overall average granularity of a component must follow a normal distribution (with a given mean and

variance) as per statistical laws. This requirement is necessary in order to ease implementation of statistical and mathematical operations required during analysis. The properties of Normal distributions are very well understood. Algebraic and statistical operations can be easily performed using data that is normally distributed.

Note that the granularity estimation approach described in Section 5.1.2, which essentially computes the average of average granularity values, always yields a normally distributed average statistic [34]. However, sufficient random samples must be used to ensure that a representative distribution is computed. For this part, we assume that the modeler provides a sufficient number of samples and verifies the normality of data using the statistical techniques described in Section 5.1.2.

2. It has been assumed that the typical events of a component can be identified and the TEG granularity of components can be estimated. We assume that the modeler appropriately specifies the set of events that must be used to compute the TEG of each component. This is an important aspect; if a component's TEG is not accurately determined then DCSPPM estimates will demonstrate large errors.

3. It is assumed that the variances in TEG are small, typically of the order tens of microseconds. This assumption is necessary because variance increases as a square of the estimates, unlike the mean value which increases linearly. Consequently, when variance is large the gross estimates may include variances that are larger than the mean, thereby rendering the estimate practically useless. For example, if the variance is large, the DCSPPM may predict the change due to DCS to be 5($\pm$ 10%). In other words, the change could be in the range –5% (decrease in simulation time) to +15% (increase in simulation time). Such an estimate provides no useful information to the user.

4. In other words, it is assumed that the granularity of each component does not significantly skew during simulation. Furthermore, the output probabilities of the components do not skew during simulation, i.e. each component generates output events as predicted during static analysis.

5. Currently, DCSPPM does not handle loops in the model and arbitrarily breaks them. It is assumed that loops in the model do not introduce an excessive number of additional events which currently are not predicted by static analysis.

6. It is assumed that the underlying simulation kernel scales linearly with respect to the number of events and simulation objects. Based on this assumption, linear arithmetic operations have been used in the implementation of DCSPPM. This is an important assumption because the characteristics of components are estimated using small, fast simulations which involve very few components. On the other hand, the final models may contain hundreds or thousands of components. If the simulation kernel does not scale linearly, linear operations on granularity estimates will no longer be valid.

7. The simulation platform is assumed to be homogeneous in workstation and network configurations. Each workstation is assumed to have the same hardware and software characteristics. Furthermore, the network connections between each pair of workstations is assumed to have identical latencies and bandwidth.

8. It is assumed that the workload on the workstations does not significantly change during simulation. The workload on the workstations directly influences the granularity of the components simulated on the workstations. If the load on one of the workstations changes, then the characteristics of the simulation (particularly parallel simulation) may change significantly. Consequently, the estimates generated by DCSPPM will no longer be valid. However, it must be noted that, in general, dynamic changes to the simulation environment cannot be tracked by a static estimation approach such as DCSPPM.

9. It is assumed that the network characteristics do not significantly skew due to external loads during simulation. If there is significant external load on the network, it impacts parallel simulation performance. Therefore, DCSPPM predictions will no longer be accurate.

6. Empirical Evaluation of DCSPPM

Empirical evaluation of DCSPPM was primarily conducted to evaluate the accuracy of the performance estimates generated by the proposed methodology. For this purpose, a variety of models undergoing both positive (i.e. abstractions that improve performance) and negative (i.e. abstractions that decrease performance) were used. An overview of the various models used for experiments is described in Section 6.1. The results from these experiments, conducted using a varying number of workstations, are discussed in Section 6.2. Furthermore, additional experiments were conducted to estimate sensitivity of the DCSPPM predictions to various extrinsic factors such as prolonged fluctuations in CPU load, changes in network traffic and GVT-based garbage collection rates. Recall that these factors are assumptions underlying the design

Table 3. Characteristics of models

Model	Number of components			
Name	Atomic	Abstract	Others	Total
Adder	482	96	2	580
Multiplier	16360	3096	2	19458
pASIC	825	25	2	852
ATM-Net	126	3	0	129
MANET	88	4	0	92

and implementation of DCSPPM (see Section 5.5). Experiments related to sensitivity analysis of DCSPPM are presented in Section 6.5.

6.1 Models

The models used in the empirical evaluation were borrowed from studies conducted to evaluate the applicability of dynamic resolution changes in parallel simulation. In other words, realistic models that use multi-resolution modeling for different tasks were used for evaluating the effectiveness of DCSPPM. Some of the salient characteristics of the models used in the experiments are listed in Table 3.

All of the models were described in a hierarchical fashion using SSL (WESE's frontend modeling language) by suitably interconnecting atomic components developed using WESE's API. The atomic components are at the highest level of resolution and cannot be refined any further. The number of atomic components in the models at the least abstract configuration is shown in the column headed Atomic in Table 3. The SSL description also included a number of abstract components that represent the low-resolution equivalent for a given set of atomic components. Depending on the model, the abstract components replaced 4–12 atomic components. The column headed Abstract in Table 3 indicates the number of such abstract components in each model. These models also included components that generated primary inputs and captured outputs. The number of such auxiliary components in each model is shown in the column headed Others in Table 3.

As tabulated in Table 3, the models used in the experiments can be classified into the following three main categories.

1. Logic Circuit Models: The first three models shown in Table 3, namely Adder, Multiplier and pASIC, were digital logic circuits. They were primarily designed to demonstrate the effectiveness of applying DCS-based dynamic, multi-resolution models for VLSI power estimation [1, 8]. The circuits were modeled in a hierarchical fashion using basic logic gates such as AND, OR and NOT gates. The column headed Atomic in Table 3 indicates the number of such components used for modeling. The models also included more abstract logic components such as exclusive-OR gates and FullAdder components. The column headed Abstract in Table 3 indicates the number of such abstract components.

   The Adder models a 32-bit ripple carry adder [36]. It does not have any loops and experiences negligible rollbacks. It can be considered as an ideal candidate for DCSPPM.

   The Multiplier is a large model with complex interconnections between components. Although this model does not have loops in it, the inherent design causes numerous rollbacks and is a stress test for both Time Warp and DCSPPM [1, 8].

   The pASIC model has numerous loops in it and does not have a deep hierarchical organization. Unlike the earlier models, this model requires two-phase analysis to resolve the loops. However, the loops do not give rise to oscillations in this model.

   All of the digital logic models have few I/O ports and, as described in Section 5.1.1, the behavior of the components can be readily collated and used. DCSPPM was used to predict performance changes of abstracting several parts of these models.

2. ATM Network Models: The ATM-NET model is a detailed, cell-level model of an Asynchronous Transfer Mode (ATM) network. It utilizes the Private Network-to-Network Interface (PNNI) signaling and control protocol that provides scalable, dynamic QoS-based link-state routing. The model was originally developed to explore applicability of dynamic DCS-based multi-resolution models to accelerate simulations of rare event scenarios [1, 25]. The ATM-NET model had nine ATM switches organized into three hierarchical ATM clouds. An ATM cloud is an abstraction of a given number of ATM switches. DCSPPM was used to analyze the performance impact of abstracting three ATM switches using an ATM cloud. The ATM switches and cloud components use a varying number of I/O ports. Consequently, they require on-demand generation of behavior tables. This model exercises parts of the DCSPPM implementation that earlier models did not.

3. Mobile Ad hoc Networks (MANET) Models: The last model shown in Table 3 is a spatially explicit model of a Mobile Ad Hoc Network (MANET) used for an asset tracking system [1]. The model involves 80 mobile assets tracked by 8 fixed-base stations using ad hoc networking techniques. Dynamic Source Routing (DSR) protocol has been employed for ad hoc packet routing. All communication messages are routed to other assets via a hierarchical area composed of sub-area components. The spatially-explicit hierarchical area is aggregated or

divided using DCS into larger or smaller units, depending on the overlap of the communication range of the wireless assets. The objective is to minimize simulation overhead of routing packets by dynamically adapting the logical partition of overlapping wireless assets [1]. This model is a stress test for the accuracy of DCSPPM because the MANET models do not have a fixed communication pattern. However, since the number of mobile entities are fixed and the area in which they move is fixed, there are bounds on the model's characteristics which are captured using larger variances.

6.2 Experiments

The experimental evaluation of DCSPPM was conducted using the models described in Section 6.1. The experimental platform consisted of a set of networked workstations. Each workstation consisted of two Athlon processors (1 GHz) with 1 Gb of main memory running Linux (kernel 2.4.2). The workstations were networked using Gigabit Ethernet. The empirical evaluation of DCSPPM was performed in the following manner.

A set of transformations were chosen for each model. DCSPPM was used to estimate the change in simulation time due to the selected set of transformations. The model was then simulated with and without the abstractions for sufficient duration (or with sufficient number of input vectors) and the time for simulation was compared. The observed change in simulation time was then statistically compared with the estimate generated by DCSPPM.

The statistics collated from the experiments are shown in Table 4. The columns headed #DCS, #CPUs, Estimate, Observed and Error indicate the number of abstractions, number of CPUs (or WESE factories) used for simulation, DCSPPM estimate of change in performance, observed change in performance, and the percentage error between the estimate and observation, respectively.

As illustrated by the final column in Table 4, DCSPPM generates good estimates of the change in performance due to abstraction of parts of a model using DCS. Positive estimates indicate improvement in performance or decrease in simulation time when abstractions are applied. Conversely, negative estimates indicate decrease in performance. For example, in the case of the Multiplier model, a set of four modules were abstracted at different spots in the model in the two different cases shown in Table 4. Performance improves in one case while performance degrades in another, highlighting the dilemma involved in using multi-resolution models.

The time durations for which the models were simulated was set by trial-and-error to the shortest duration after which valid performance comparisons could be made. The graph in Figure 11 shows the percentage change in simulation time due to a given abstraction in the various models. As indicated in the graph, different models require a different number of input vectors in order to obtain stable, average observations. The experimental observations in Table 4 were made at the knee point in the curves indicated by gray circles in Figure 11. The graph also highlights the issues involved in using trial-and-error experiments (see Section 1) to determine the impact of a DCS transformation.

Table 4. Statistics from experiments

Model (#DCS)	#CPU	Estimate (%)	Observed (%)	Error (%)
Adder (4)	1	10.95 ± 0.01	10.966	0
	2	19.77 ± 0.8	23.06	2.49
	6	17.47 ± 1	17.35	0
Adder (1)	1	–4.52 ± 0.01	–4.87	0.35
	2	–5.62 ± 0.72	–5.02	0.6
	6	–7.85 ± 1.68	–6.75	1.1
Mul32 (4)	1	–3.08	–3.06	0.02
	2	–6.72 ± 0.03	–8.64	1.92
	6	–4.27 ± 0.5	–4.256	0
Mul32 (8)	1	3.9	4.87	0.97
	2	2.81 ± 0.04	4.64	1.83
	6	6.9 ± 1.2	5.13	0.57
pASIC (8)	1	–1.27	–1.09	0.175
	6	–4.53	–4.10	0.43
pASIC (12)	1	4.38	3.41	0.97
	6	3.21 ± 0.08	3.64	0.35
ATM-Net (1)	1	15.27 ± 0.01	14.54	0.73
	4	10.23 ± 0.97	10.26	0
ATM-Net (3)	1	41.58 ± 0.01	40.8	0.78
	4	30.35 ± 1.22	30.16	0
	6	30.35 ± 1.22	30.16	0
MANET (4)	1	59.95	59.89	0.05
	2	17.47 ± 1.45	17.0	0.47
	4	5.4 ± 2.07	5.47	0
	6	3.35 ± 2.57	3.5	0

6.3 DCSPPM Timings

The time taken for performing various phases in DCSPPM is tabulated in Table 5. The time for the fastest running test simulation is also shown for comparison. Note that the granularity estimation and communication latency measurement is a one-time task. On the other hand, DCSPPM analysis may be repeated several times for different combinations. As illustrated in Table 5, DCSPPM analysis phase runs orders of magnitude faster than the fastest test simulation. As illustrated by the experiments, DCSPPM provides a rapid approach for estimating the performance impacts of DCS transformations, thereby providing a more scientific approach to enabling effective use of multi-resolution parallel simulations.

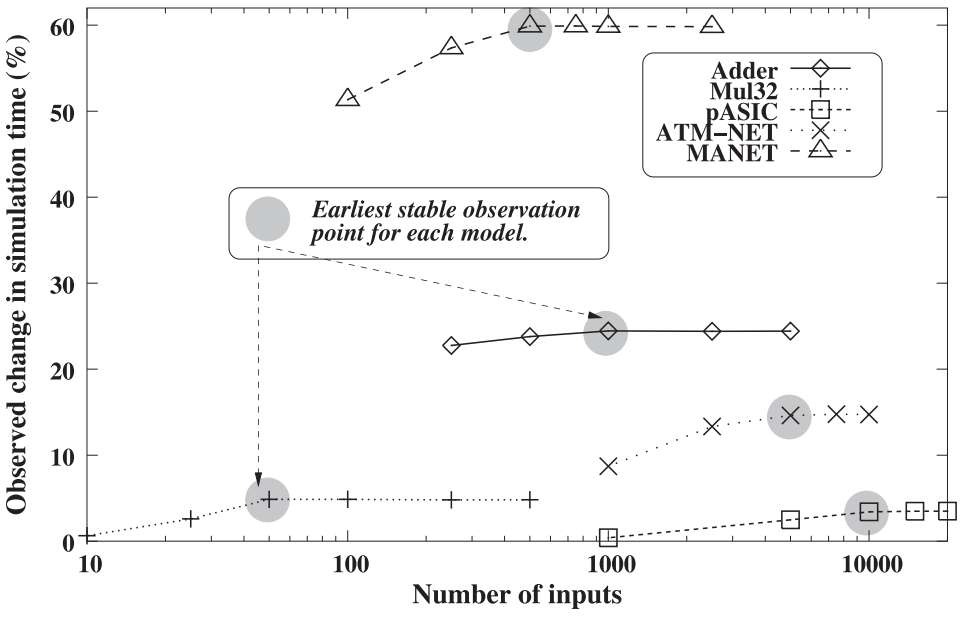


Figure 11. Observation points for performance comparisons

Table 5. Time for DCSPPM versus simulation

Model	Time (s)			
	Granularity estimate	Communications latency	DCSPPM analysis	Fastest test
Adder	6.23	15.48	0.457	36.86
Mul32	6.23	15.48	4.73	242.8
pASIC	6.23	15.48	3.72	45.83
ATM-Net	11.23	15.75	0.031	27.5
MANET	2.23	15.42	0.0025	11.93

6.4 Sources of errors

The estimates generated by DCSPPM have some errors in them, as indicated in Table 4. The source of errors include minor skew in model behavior, nonlinearities in the simulation kernel, changes in characteristics of the communication network, operating system activities and deviations in model behavior. Conspicuous nonlinear performance characteristics were observed when the simulations exceeded main memory (RAM) limits on the workstations. The nonlinearities are attributed to virtual memory overheads, in particular due to swapping data back-and-forth from the hard disk drive. Where virtual memory sizes were exceeded, there were significant nonlinearities and estimates generated by DCSPPM demonstrated notable errors; DCSPPM cannot be used in such cases. Other dominant sources of errors in our experiments include the following.

- Nonlinearities in the simulation kernel: The dominant source of nonlinearity in the kernel arises from Global Virtual Time (GVT) computations and garbage collection overheads that are necessary in a Time Warp simulation. The graph in Figure 12(a) illustrates the impact of GVT on simulation time. The data shown in the figure is the simulation time of the Adder model using 1000 input vectors approximated using the Bezier curve-fitting algorithm. It was noted that if the GVT period is not set in the linear region, then the observations from simulations were skewed.
- Nonlinearities in the network: The other dominant source of error arises from the underlying Gigabit network. The nonlinearities are conspicuous due to bursty communication behavior of a Time Warp simulations, particularly during rollbacks. The graph in Figure 12(b) illustrates the average message latency with different burst sizes. As illustrated by the graph, the average latency significantly skews depending on the total number of messages exchanged. This behavior skews the sim-

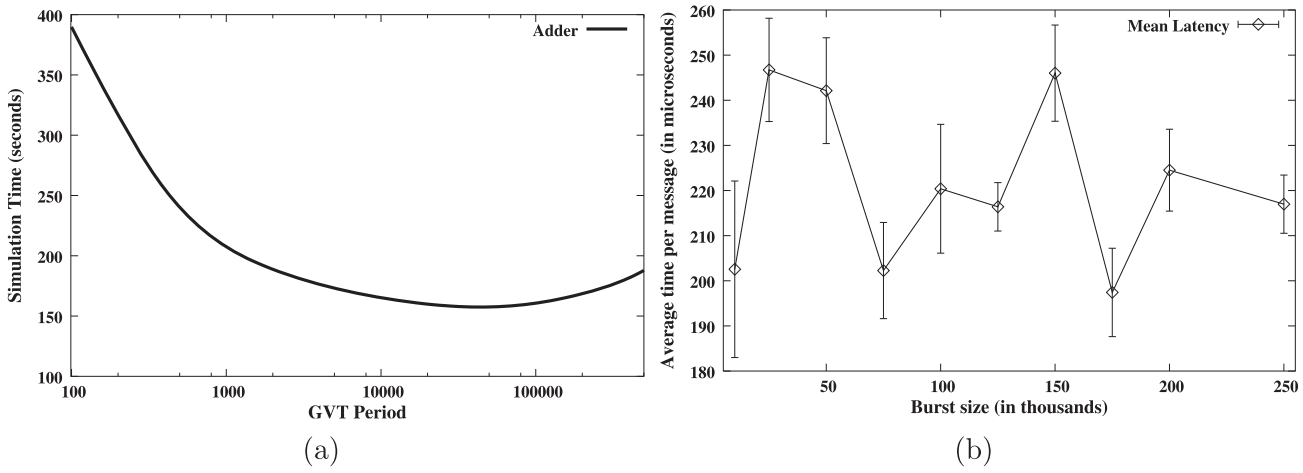


Figure 12. Sources of errors: (a) GVT period impact and (b) network latency

ulations and the DCSPPM estimates, introducing errors. Such nonlinearities cannot be fully accommodated in a static technique such as DCSPPM. Attempting to incorporate the extreme cases in the estimation technique significantly increases the variance in the reported data, reducing the practical usefulness of DCSPPM's estimates. The inherent nonlinearity of the communication sub-system impacts the overall accuracy of the estimates generated by DCSPPM.

6.5 Sensitivity Analysis

The time taken to simulate a model is strongly influenced by the simulation platform characteristics, which impact the estimates generated by DCSPPM. In this section, we explore the sensitivity of the DCSPPM to changes in the configuration of the simulator and simulation platform. These experiments cross-validate the assumptions underlying DCSPPM. Note that sensitivity to virtual memory usage was not analyzed because DCSPPM estimates were found to be extremely sensitive to this parameter. The estimates were impacted because of nonlinearities in virtual memory overheads, such as disk swapping.

The sensitivity of DCSPPM to GVT period, extraneous CPU load and extraneous network load are shown by the graphs in Figure 13. The sensitivity experiments were conduced using parallel simulations conducted on two workstations. These experiments were conducted using the Multiplier model because it is a large and complex model and change in characteristics were readily observable. As illustrated by Figure 13(a), the estimates are not sensitive to the GVT period in the linear region of operation. Recall that GVT period introduces some nonlinearities in the simulation kernel (see Section 6.4 for details) because the GVT period affects simulations with and without DCS in an identical manner. Consequently, the change in the simulation time is almost constant.

The graph in Figure 13(b) illustrates the deviations due to extraneous network load. The extraneous network load on each workstation was generated using an external client-server program that exchanges data at a fixed rate. As shown by the graph, the estimates are sensitive to network load. However, the deviations are also influenced by the nonlinear characteristics of the network. The impact of extraneous CPU load on the estimates are shown by Figure 13(c). The CPU load was generated by running an additional process on each workstation that performed a finite amount of mathematical computations, thereby consuming CPU time. The volume of processing was varied to generate different loads on the CPU. As illustrated by Figure 13(c), the estimates are very sensitive to CPU load. In other words, DCSPPM estimates are least sensitive to GVT period changes but are more sensitive to changes in network and CPU load.

7. Conclusions

Selective, dynamic abstraction and refinement of multiresolution models via Dynamic Component Substitution (DCS) is a powerful technique to accelerate various simulations. However, care must be taken to ensure that model transformations do not have negative impacts. Consequently, the impact of transformations must be know prior to simulation. In an attempt to address the issue of selecting suitable DCS transformations, a *DCS Performance Prediction Methodology* (DCSPPM) was proposed. The issues involved in the design and implementation of DCSPPM were presented in this article. Empirical evaluation of DCSPPM using diverse models was discussed.

The experiments indicate that the DCSPPM generates good estimates with errors less than $\pm 3\%$ in our experi-

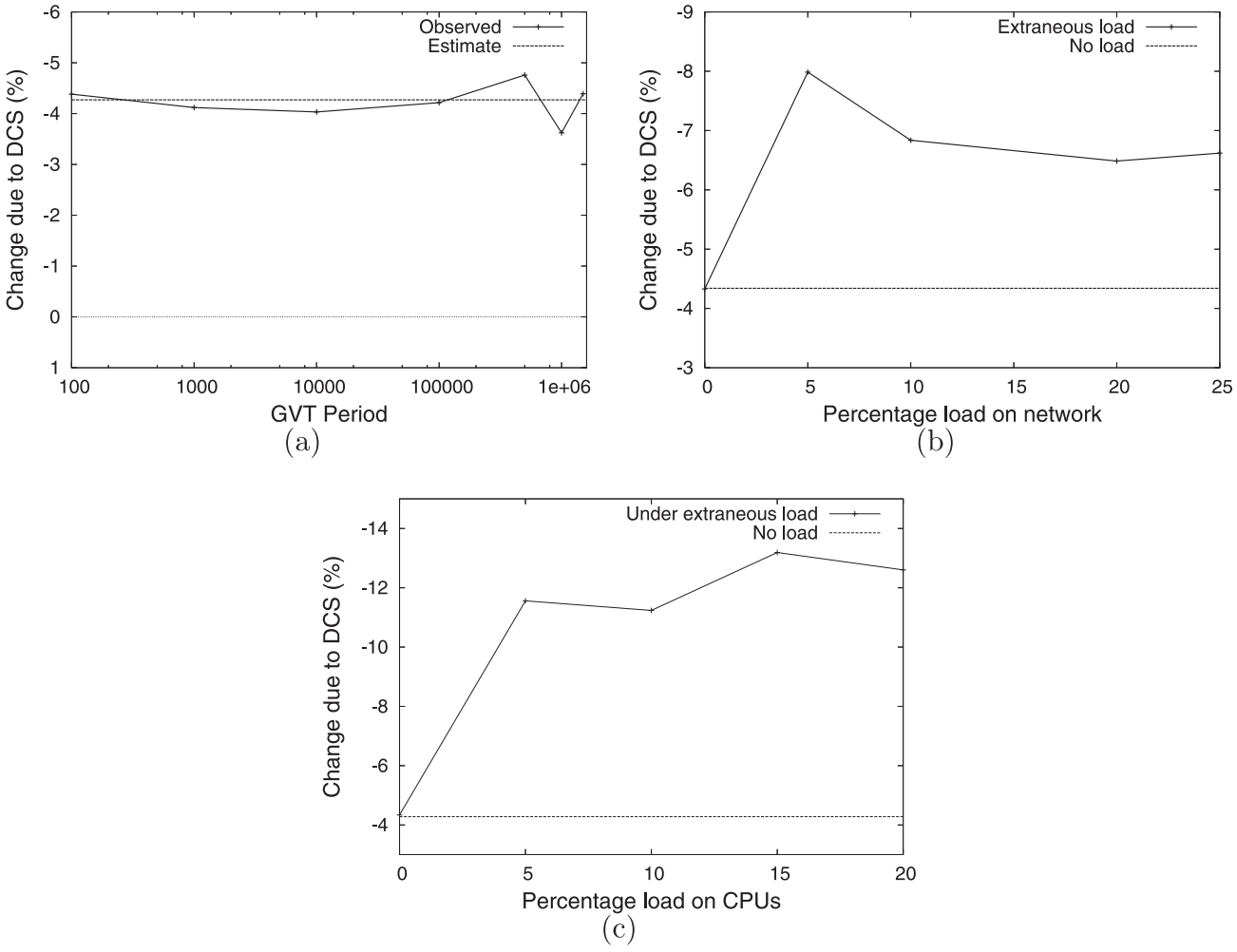


Figure 13. Sensitivity analysis of DCSPPM to extraneous loads: (a) GVT period; (b) network load; and (c) CPU load

ments. Sources of errors in the estimates were presented and their influences were empirically explored. DCSPPM estimates were shown to be valid as long as the characteristics of the model and simulation platform do not significantly skew during simulation. The sensitivity of DCSPPM to external factors was also empirically explored. The article pitched the timing for DCSPPM analysis against the shortest possible parallel simulation time for different models to highlight the speed of DCSPPM. Since DCSPPM runs very fast, it can be used to explore numerous model configurations to identify the most optimal candidate for a given analysis. It must be noted that DCSPPM aims to identify the best combination of transformations given a set of choices, and not the absolute optimal configuration for a given model.

The experiments presented in this article show that the estimates generated by DCSPPM closely track the observed changes, highlighting the effectiveness of DCSPPM. The estimates may also be used as indicators for further model development and refinement. A number of additional optimizations can be implemented to further reduce the analysis time and improve the accuracy of the estimates. We are continuing our pursuit to enhance DCSPPM and apply it to diverse problem domains and synchronization methodologies. Our studies strongly indicate that DCS, coupled with DCSPPM, is is an effective methodology to enable efficient use of multi-resolution simulations. Furthermore, a modeler can utilize DCSPPM estimates to intelligently fine-tune the models to achieve maximum efficiency. We believe DCSPPM paves the way to replacing the estimations involved in effective use of parallel multi-resolution simulations with a robust and easy to use systematic scientific methodology.

8. References

[1] Rao, D. M. 2003. *Study of Dynamic Component Substitution*. Ph.D. thesis, University of Cincinnati.

[2] Wilsey, P. A. 2000. Modeling, analysis and simulation of computer and telecommunication systems. *In* A. Kent, (Ed.) *Encyclopedia of Library and Information*. Marcel Dekker, Inc.

[3] Fishwick, P. A. 1995. *Simulation Model Design and Execution: Building Digital Worlds*. Englewood Cliffs, NJ: Prentice Hall.

[4] Paxson, V. and S. Floyd. 1997. Why we don't know how to simulate the internet. *In Proceedings of the 1997 Winter Simulation Conference (WSC'97)*, pp. 44–50.

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

[6] Ashenden, P. J. 2001. *The Designers Guide to VHDL. Second Edition*. San Mateo, CA: Morgan Kaufmann Publishers Inc.

[7] 2002. *IEEE Standard VHDL Language Reference Manual*. 3 Park Avenue, New York, NY 10016-5997, USA.

[8] Rao, D. M. and P. A. Wilsey. 2006. Predicting performance of resolution changes in parallel simulations. *In Proceedings of the 20th Workshop on Parallel and Distributed Simulation (PADS'06)*, pp. 45–54. Singapore.

[9] Perros, H. G. 2005. *Connection-oriented Networks: SONET/SDH, ATM, MPLS and Optical Networks*. NJ, US: John Wiley & Sons Ltd.

[10] Fujimoto, R. M. 1993. Parallel discrete event simulation: Will the field survive? *ORSA Journal on Computing* 5(3).

[11] Ferscha, A. and J. Johnson. 1996. A testbed for parallel simulation performance predictions. *In 1996 Winter Simulation Conference Proceedings*.

[12] Jain, R. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY: Wiley-Interscience.

[13] Dickens, P., M. Haines, P. Mehrotra, and D. Nicol. 1996. Towards a thread-based parallel direct execution simulator. *In Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS-29)*, pp. 424–433.

[14] Balakrishnan, V., P. Frey, N. Abu-Ghazaleh, and P. A. Wilsey. 1997. A framework for performance analysis of parallel discrete event simulators. *In Proceedings of the 1997 Winter Simulation Conference*.

[15] Liu, J., D. M. Nicol, B. J. Premore, and A. L. Poplawski. 1999. Performance prediction of a parallel simulator. *In Workshop on Parallel and Distributed Simulation*, pp. 156–164.

[16] Gupta, A., I. F. Akyildiz, and R. M. Fujimoto. 1991. Performance analysis of Time Warp with multiple homogeneous processors. *IEEE Transactions on Software Engineering* 17(10), 1013–1027.

[17] Tay, S. C., Y. M. Teo, and R. Ayani. 1998. Performance analysis of time warp simulation with cascading rollbacks. *In Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, pp. 30–37.

[18] Perumalla, K. S., R. M. Fujimoto, P. J. Thakare, S. Pande, H. Karimabadi, Y. Omelchenko, and J. Driscoll. 2006. Performance prediction of large-scale parallel discrete event models of physical systems. *In Proceedings of the 2006 Winter Simulation Conference (WSC'06)*, pp. 356–364.

[19] Rao, D. M., V. Chernyakhovsky, and P. A. Wilsey. 2000. WESE: A Web-based Environment for Systems Engineering. *In Proceedings of International Conference On Web-Based Modelling & Simulation (WebSim'2000)*. Society for Computer Simulation.

[20] Rao, D. M., R. Radhakrishnan, and P. A. Wilsey. 1999. Web-based network analysis and design. *ACM Transactions on Modeling and Computer Simulation* (forthcoming).

[21] Huang, P., D. Estrin, and J. Heidemann. 1998. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. *In Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Networks*.

[22] McBrayer, T. and P. A. Wilsey. 1995. Process combination to increase event granularity in parallel logic simulation. *In 9th International Parallel Processing Symposium*, pp. 572–578.

[23] Rao, D. M. and P. A. Wilsey. 2006. Applying parallel, dynamic-resolution simulations to accelerate vlsi power estimation. *In Proceedings of the 2006 Winter Simulation Conference (WSC'06)*. (forthcoming).

[24] Rao, D. M. and P. A. Wilsey. 2006. Accelerating ATM simulations using dynamic component substitution (DCS). *Simulation* (forthcoming).

[25] Rao, D. M. and P. A. Wilsey. 2005. Accelerating spatially explicit simulations of spread of lyme disease. *In Proceedings of the 38th Annual Simulation Symposium*, pp. 251–258. San Diego, California, USA.

[26] Reynolds, P. F., A. Natrajan, and S. Srinivasan. 1997. Consistency maintenance in multiresolution simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 7(3), 386–392.

[27] Rao, D. M., P. A. Wilsey, and H. W. Carter. 2001. Optimizing costs of web-based modeling and simulation. *In Proceedings of the First International Workshop on Internet Computing and E-Commerce (ICEC'01)*. IPDPS.

[28] Tremblay, J. P. and R. Manhohar. 1975. *Discrete Mathematical Structures With Applications to Computer Science*. US: McGraw-Hill Computer Science Series.

[29] Rao, D. M. and P. A. Wilsey. 2000. Dynamic component substitution in web-based simulation. *In Proceedings of the 2000 Winter Simulation Conference (WSC'2000)*. Society for Computer Simulation.

[30] Rao, D. M. and P. A. Wilsey. 2002. Performance prediction of dynamic component substitutions. *In Proceedings of the 2002 Winter Simulation Conference (WSC'02)*.

[31] Radhakrishnan, R., D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. 1998. An Object-Oriented Time Warp Simulation Kernel. *In* D. Caromel, R. R. Oldehoeft, and M. Tholburn, (Eds.) *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, volume LNCS 1505, pp. 13–23. Springer-Verlag.

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

[33] Rao, D. M., N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. 1998. Unsynchronized parallel discrete event simulation. *In Proceedings of the 1998 Winter Simulation Conference*, pp. 1563–1570.

[34] Hogg, R. V. and A. T. Craig. 1995. *Introduction to Mathematical Statistics*. Englewood Cliffs, New Jersey: Prentice Hall.

[35] 2006. *An Introduction to R, A Programming Environment for Data Analysis and Graphics*. (online at http://www.r-project.org/).

[36] Patterson, D. A. and J. L. Hennessy. 2005. *Computer Organization And Design: The Hardware/Software Interface. Third Edition*. San Francisco: Elsevier Inc.