Predicting Performance of 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.
philip.wilsey@uc.edu

Abstract

*Multi-resolution models can be statically (*i.e., *before simulation) or dynamically (*i.e., *during simulation) abstracted to accelerate the simulations without compromising the analysis goals. However, abstractions must be carefully chosen because not all abstractions improve performance. Unfortunately, identifying performance enhancing transformations, particularly in parallel simulations, is a complex task. We have been investigating this bottleneck using a component-based, Time Warp synchronized modeling and simulation environment called* WESE *in which static and dynamic abstractions are performed using a methodology called Dynamic Component Substitution (DCS). Our ongoing research has resulted in further advances to a novel DCS Performance Prediction Methodology (DCS*PPM*). DCS*PPM *combines and compares platform-specific performance characteristics of components via static analysis of the model to predict performance changes due to DCS transformations. This paper presents the enhanced version of DCS*PPM *and experiments. Empirical evidence obtained using diverse models indicates that DCS*PPM *provides good estimates (error* $< 5\%$*) of the performance impacts of DCS transformations.*

1 Introduction

Simulation of large, high resolution models is a time consuming task even when parallel simulation techniques are employed [6]. Furthermore, if analysis deals with specific subsystems or selected scenarios, processing voluminous data from inconsequential scenarios exacerbates analysis [6]. Consequently, multi-resolution modeling and simulation techniques are used to obtain more optimal tradeoffs between analysis requirements, accuracy, fidelity, and performance [6, 10]. In our research, we have enabled multi-resolution simulation using hierarchical, component-based models and a novel methodology called Dynamic Component Substitution (DCS) [6]. In DCS a set of components called a *module* is substituted with an equivalent component or vice versa to enable abstraction and refinement [6]. DCS transformations may be performed statically (*i.e.,* prior to simulation) or dynamically (*i.e.,* during simulation) to change the resolution of the model [6]. A more detailed overview of DCS is presented in Section 2.

Typically, abstractions are used to improve simulation performance while trading-off observability and possibly some fidelity. Unfortunately, in many scenarios abstraction negatively impacts performance and simulation times increases [6]. In other words, DCS transformations, to parts of a model, that seemingly improve performance have counterintuitive impacts on the model as a whole. Several examples in which abstraction deteriorates performance are discussed in the literature [6]. In addition, similar transformations to different parts of a model have different and often converse results [6]. Consequently, the effects of a transformation on simulation performance must be known apriori in order to enable effective use multi-resolution models. A conventional approach to determine the impacts is to use short test simulations. However, numerous test simulations need to be performed for each transformation to obtain reliable estimates. The scenario is exponentially complicated in the case of parallel simulation that are conducted using varying number of workstations.

We have been investigating approaches to predict performance impacts of DCS transformation and ease effective use of multi-resolution models. Initially, we explored the use of a more parsimonious approach for performance prediction [8]. However, the earlier approach had significant errors (sometimes $> 50\%$) and could not be effectively used. Our continued research have resulted in the development of a DCS Performance Prediction Methodology (DCSPPM) that uses simulation-platform specific estimates and static analysis of a model

Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)
0-7695-2587-3/06 $20.00 © 2006 IEEE

to predict the performance impacts of DCS transformations. For example, given two DCS transformations, say τ_1 and τ_2, DCSPPM yields quantitative estimates, say $4.5 \pm 1\%$ and $-2.3 \pm 0.5\%$. 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. Note that several studies related to performance prediction of parallel simulations have been reported [2, 3, 4]. Unlike earlier investigations, the focus of this study is to predict the change in performance due to resolution changes rather than the performance of the simulation as a whole.

This paper presents the latest results and numerous enhancements to DCSPPM conducted over a period of two years after our earlier publication [8]. A brief background on WESE and DCS is presented in Section 2. A detailed description of DCSPPM is presented in Section 3. The experiments conducted to evaluate the accuracy of DCSPPM are presented in Section 4. Section 5 concludes the paper summarizing contributions from this work.

2 Background

The performance prediction algorithm proposed in this paper has been implemented as part of a component-based modeling and simulation environment called WESE. WESE is an acronym for Web-based Environment For System Engineering. Model development in WESE involves two phases. First, a set of components involved in the model are developed using WESE's Application Program Interface (API) and bundled into a WESE factory. A WESE factory is a repository of components with additional capability for parallel simulation. Each component is associated with a *stub* object that is responsible for creating the object on demand. In addition, the stubs have also been used to store behavior and performance characteristics that play a central role in the proposed methodology. The WESE factories are deployed on various workstations to be used for parallel simulation. Interactions with a factory are performed via TCP/IP sockets using a custom protocol. Next, multi-resolution models are developed by suitably interconnecting a set of modules via ports that constitute the interface to a module. A module is a set of port-interconnected components (present on a given factory) and submodules. It also has a more abstract, Equivalent Component (EC) associated with it. An EC has an identical interface and similar functionality (as defined by the modeler) as that of the corresponding module. The model includes all the information and parameters to setup a parallel simulation using a given set of WESE factories.

Abstraction or refinement of the model is implemented by substituting a module with its EC or vice versa using a methodology called Dynamic Component Substitution (DCS) [6]. Substituting a module with its equivalent component is synonymous to abstraction. DCS also involves appropriately mapping the states of the components participating in the transformation. Strategies used for triggering DCS during simulation are broadly categorized into *proactive* (DCS scheduled in future) and *reactive* (scheduled in the recent past) strategies. A combination of proactive and reactive DCS transformation have been employed to accelerate parallel simulations in diverse applications such as: logic simulation, VLSI power estimation, rare phenomena simulation, mobile ad hoc network simulations, and simulation of Eco systems. The core constructs and several optimizations underlying DCS have been rigorously defined using a discrete mathematics algebra. DCS provides a flexible, scalable, and performance-predictable strategy for enabling dynamic multi-resolution simulations thereby improving overall efficiency of simulation studies [6]. A detailed description on DCS and its aforementioned usage is available in the literature [6].

In WESE, an event-driven approach has been utilized to sequence the various phases involved in enabling DCS. The event-driven approach has been developed by extending the backbone provided by the underlying simulation kernel called WARPED. WARPED is a general purpose, Time Warp synchronized parallel simulation kernel. WESE leverages the Time Warp infrastructure of WARPED in an unique manner to enable proactive and reactive DCS transformations. WESE also provides an API for mapping states of components during DCS. It is the responsibility of the modeler to utilize the API and suitably map states of components during DCS transformations. Care has been taken to ensure that WESE and the model implementations preserve scalability and linearity (with respect to number of ports) to enable predictable performance characteristics. A more detailed description of WESE, WARPED, and DCS implementation are available in the literature [5, 7, 9].

3 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 of the DCSPPM is shown in Figure 1. 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 the Figure 1, DCSPPM proceeds in four distinct but overlapping phases. A discussion of the four phases is presented in the following subsections.

3.1 Phase 1: Collating behavior tables for components

The first phase of DCSPPM involves collating the data for generating the Behavior Table (BT) for each component. A BT indicates the input-output characteristics of the component. Each row specifies the probability of output vectors (or events) being generated at each output port for a given combination of input vectors (or events) at a component's input ports. Components that do not have any inputs or outputs are treated as a special case and are described using a NULL input or output vectors. The set of I/O vectors are implicitly ordered to reflect the logical port numbers of a component to minimize size of BTs. Each BT entry contains a set of I/O vectors that are grouped based on simulation timestamp values. The I/O vectors at each port is represented using a 3-tuple consisting of $< I/OProbability, RealTime, Factory - ID >$, where

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 maybe empirically determined using test simulations (see Figure 1). Such test simulations are effective for components with few input and output ports (2 to 5 ports). All possible combinations of input vectors are fed to the component and the resulting output is analyzed to determine I/O probability. However, such an approach does not scale for large components. In such cases, the BT entries are computed on-the-fly as needed. In this case, the stubs associated with the components, provides BT entries on demand, once the actual input vectors to be analyzed is known. Furthermore, it computes output probabilities based on model specific knowledge provided by the modeler.

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 1 indicates that if an input vector is presented to the component at real time 0, it generates output at 4.75 ± 0.1 μsec. This value 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. Typical Event Granularity (TEG) is assumed to follow a Normal distribution in concordance with statistical theories [1]. Prior to usage of DCSPPM, it is suggested that the assumption of normality be verified [1] as illustrated in the literature [6]. The primary motivation for normality verification is to ensure that sufficient number of samples have been collated to yield a good representative statistic.

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 requires communication over the network. Tracking such network centric communication points serves the following two purposes in DCSPPM: *(i)* It enables appropriate inclusion of communication latencies that impact the overall TEG of a model; and *(ii)* it is used to identify points where potential for rollbacks exist in order to account for synchronization overheads in Time Warp simulations.

3.2 Phase 2: Estimating Network Latencies

Estimation of communication latencies is performed by a pair of WESE factories. 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 for the messages. A number of the round trip times are measured and 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 Figure 1). The estimated values are stored in the DCSPPM Module and reused as necessary during Phase 3.

3.3 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 submodule constituting the module. The estimation is performed by propagating the BTs of components from inputs to outputs of

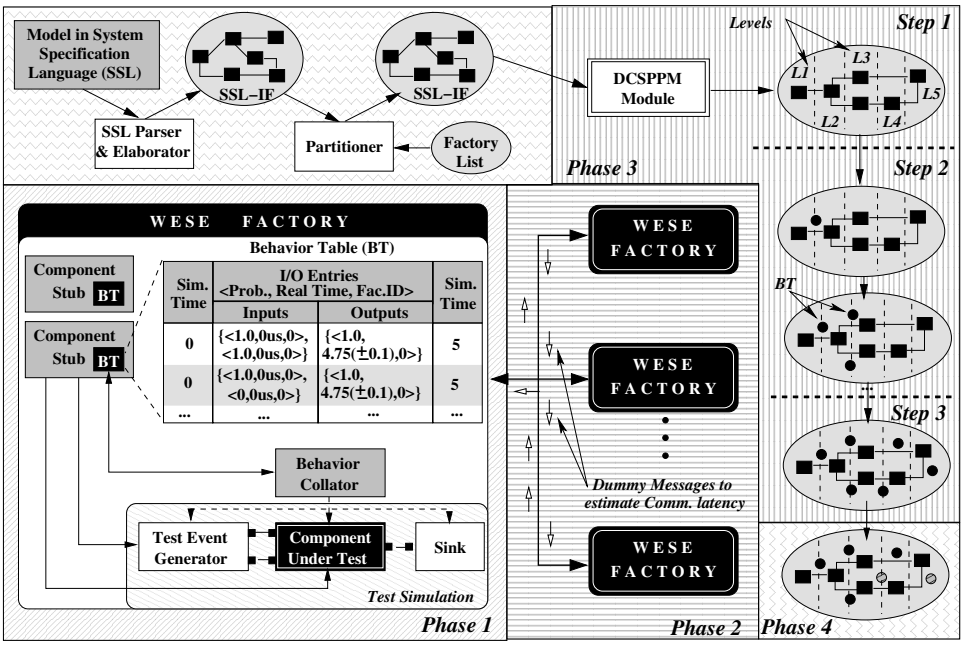


Figure 1. Overview of DCSPPM

the model. As the BTs are propagated they are suitably transformed to include the behavior and characteristics of the components constituting a module. As shown in Figure 1, this process 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 submodule is assigned a level number such that inputs of a component are at a lower level. Cycles in a module are arbitrarily resolved. Currently, DCSPPM does not effectively account for cycles in a model. Levelization results in updates and minor reorganization of the SSL-IF representation 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 IBTRs, the OBTRs are computed in the following manner:

1. First, the probabilities in an 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 an 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 component whose BT is shown in Figure 1. 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 to the IBTR simulation time to the value in the component's BT entry. In the aforementioned example, the output simulation time of the BR would be $35 + 5 = 40$.
3. Next, the maximum probability value from the IBTR is multiplied with the probability values in the component's BR entry to determine probability of OBTR. In the above example, the output probability would be: $1.0 * max(0.5, 0.75) = 0.75$.
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.
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 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 estimate the probability of rollbacks. 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 that requires additional wall clock time. Accordingly the probability of rollbacks is used to suitably scale the overall real time value of the output vectors. In addition, the rollback probabilities are propagated through the model to account for cascading rollbacks. Cascading rollbacks occur even though all events are received from the same factory because some component in the input chain rolls back.
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 for 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 weight is increased by one. The real time values for each output vector is then suitably scaled using the rollback weight value for the component, *i.e.,* $realTime = realTime + (realTime * rollback\_weight)$.
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 BR entries are statistically compared to determine rollback probability. Similar to earlier heuristic, the real time values for each output vector is suitably scaled to reflect synchronization overheads.

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 as 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 estimate 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.

3.4 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 do determine the change in performance due to a DCS transformation. Using earlier example, the difference in performance would be $(4567 \pm 38 - 4238 \pm 42)/4567 \pm 38 = 7.2 \pm 1.9\%$.

4 Experiments

The experiments conducted to evaluate the fidelity of the performance estimates generated by DCSPPM were conducted using a diverse set of models. Some of the salient characteristics of the models used in the experiments is shown in Table 1. The first three models, namely `Adder`, `Multiplier`, and `pASIC` were digital logic circuits with diverse characteristics. The circuits were modeled in a hierarchical fashion using basic logic gates such as `AND`, `OR`, and `NOT` gates. The column titled `Atomic` in Table 1 indicates the number of atomic components in the models. The models also included more abstract components such as exclusive-or gates and `FullAdder` components. The column titled `Abstract` in Table 1 indicates the number of such abstract components in a 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 titled `Others` in Table 1. The `Adder` model does not have any loops and experiences negligible rollbacks. It can be considered 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. The `pASIC` model has numerous loops in it and does not have a deep hierarchical organization.

Model Name	Number of components			
	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

Table 1. Characteristics of models

Unlike the earlier models, this model requires 2-phase analysis to resolve the loops and additional complexity to handle cascading rollbacks.

The `ATM-NET` model is a detailed, cell-level model of an Asynchronous Transfer Mode (ATM) network utilizing the Private Network-to-Network Interface (PNNI) signaling and control protocol that provides scalable, QoS-based, dynamic link-state routing. The `ATM-NET` model had 9 ATM switches organized into 3 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 last model shown in Table 1 is a spatially explicit model of a Mobile Ad Hoc Network (MANET) based asset tracking system. 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 de-aggregated using DCS into larger or smaller units depending on the overlap of the communication range of the wireless assets. The objective is to minimize the overhead of routing packets by dynamically adapting the logical partition of overlapping wireless assets. A detailed description of these models is available in the literature [6].

In concordance with the modeling strategy utilized by WESE, first a set of basic components were developed and bundled into suitable WESE factories. Next, multi-resolution models were developed by suitably interconnecting components from the appropriate factories and providing necessary parameters. The WESE factories were deployed on a dedicated network of workstations for empirical evaluation. Each workstation consisted of two Athlon processors (1 GHz) with 1 Gigabyte of main memory running Linux (kernel 2.4.2). The workstations were networked using a Gigabit Ethernet. Some of the statistics collated from the experiments is shown in Table 2. The columns titled `#DCS`, `#CPUs`, `Esti`, `Obs Change`, and `Err in Esti.` indicate the number of abstractions, number of CPUs (or WESE factories) used

Model (#DCS)	#CPU	Change due to DCS		Err In Esti.
		Esti.	Obs.	
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%

Table 2. Statistics from experiments

for simulation, DCSPPM estimate of change in performance, observed change in performance, and the percentage error between the estimate & observation respectively. The `No DCS` and `DCS` sub-columns under the `Sim Time` column indicate the time for simulating the model without any abstractions and with the indicated number of modules abstracted, respectively. Additional experimental configurations and data is available in the literature [6].

As illustrated by the last column in Table 2, DCSPPM generates good estimates of the change in performance due to abstraction of parts of a model using DCS. Refinement using DCS has the inverse effect of abstraction. 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 `Mul32` model, a set of 4 modules were abstracted at different spots in the model in the two different cases shown in Table 2. In one case, performance improves while in another case performance degrades 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 2(a) shows the percentage change in simulation time due to a given abstraction in the various models. As indicated in the graph different models

require different number of input vectors in order to obtain stable, average observations. The experimental observations in Table 2 were made at the knee point in the curves indicated by gray circles in Figure 2. This graph in Figure 2(a) also highlights the issues involved in using trial-and-error experiments to determine the impact of a DCS transformation.

As illustrated in Table 2, the estimates generated by DCSPPM have some errors. The source of errors include minor skew in model behavior, nonlinearities in the simulation kernel, changes in characteristics of the communication network, and operating system activities. Amongst the aforementioned factors, the following were found to be the most dominant ones in our experiments:

- *Nonlinearities in the simulation kernel*: The dominant source of nonlinearity in the kernel arises from Global Virtual Time (GVT) computations and fossil collection overheads that are necessary in a Time Warp simulation. The graph in Figure 2(b) 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 Bezier curve fitting algorithm. It was noted that if the GVT period is not set in the linear region, then the observations significantly skew.
- *Nonlinearities in the network*: The next dominant source of error arises from the underlying Gigabit network. The nonlinearities are conspicuous due to bursty communication behaviors of a Time Warp simulations, particularly during rollbacks. The graph in Figure 2(c) 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 in-turn skews the simulations and the DCSPPM estimates thereby introducing errors.

The accuracy of he estimates generated using DCSPPM is sensitive to the simulation platform characteristics. The DCSPPM estimates are valid as long as the load on the workstations and network do not change. The sensitivity of DCSPPM to GVT period, extraneous CPU load, and extraneous network load are shown by the graphs in Figure 3. The sensitivity experiments were conducted using the `Mul32` model because it is a large and complex model. As illustrated by the graph in Figure 3(a), the estimates are not sensitive to the GVT period which introduces some nonlinearities in the kernel. The graph in Figure 3(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 exchange data at a fixed rate. As shown by the graph, the estimates are sensitive to network load but the deviations are influenced by the nonlinear characteristics of the network. The impact of extraneous CPU load on the estimates are shown by the graph in Figure 3(c). The CPU load was generated by running an extraneous process that performed a finite amount of dummy processing thereby consuming CPU time. The volume of processing was varied to generate different loads on the CPU. As illustrated by the graph in Figure 3(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.

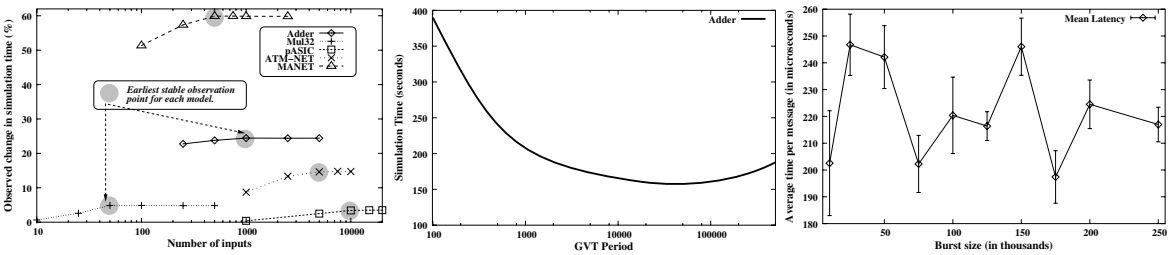
Model	Time spent for (sec)			
	Gran. Est.	Comm. Latency	DCSPPM Analysis	Fastest Test Sim.
Adder	6.23	15.48	0.457	36.86
Mul32	6.23	15.48	4.73	242.8
pASIC	6.23	15.48	3.72	45.83
ATM-Net	11.23	15.75	0.031	27.5
MANET	2.23	15.42	0.0025	11.93

Table 3. Time for DCSPPM vs. simulation

The time taken for performing various phases in DCSPPM is tabulated in Table 3. The time for the fastest running test simulation is also shown for comparisons. Note that the granularity estimation and communication latency measurement is an one time task. On the other hand, DCSPPM analysis maybe repeated several times for different combinations. As illustrated in Table 3, DCSPPM analysis phase runs orders of magnitude faster than the fastest test simulation! A much broader spectrum of experimental observations and analysis of the empirical data is available in the literature [6]. 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.

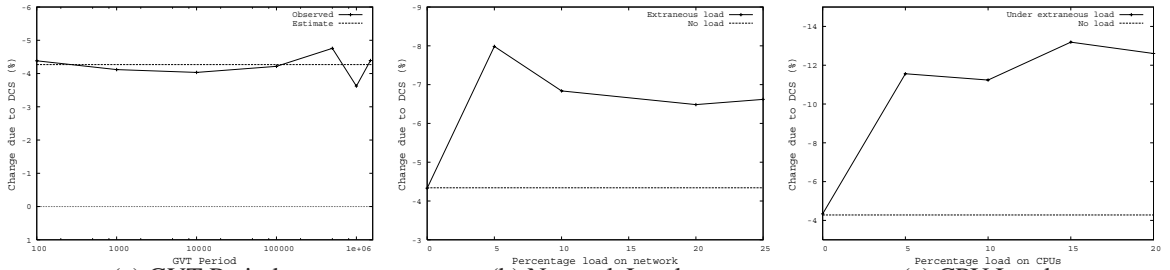
5 Conclusion

This paper presented a novel methodology called DCSPPM that can be applied to component-based, multi-resolution models to predict the performance impacts of changing the resolution using Dynamic Component Substitution (DCS). Empirical evaluation of DCSPPM conduced using diverse models were presented. The experiments indicate that the DCSPPM generates good estimates with errors less than $\pm 3\%$ in our experiments. Sources of errors in the estimates were presented and



(a) Stabilization (b) GVT Period Impact (c) Network Latency

Figure 2. Observation points and sources of errors



(a) GVT Period (b) Network Load (c) CPU Load

Figure 3. Sensitivity analysis of DCSPPM to extraneous loads

their influences were empirically explored. The estimates hold 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 presented in the paper. The paper pitched the timing for DCSPPM analysis against the shortest possible simulation time for different models to highlight the speed of DCSPPM. Since DCSPPM runs very fast, it can be used to explore numerous model configuration to identify the most optimal candidate for a given analysis. We are continuing our pursuit to enhance DCSPPM and apply it for diverse problem domains and synchronization methodologies. The objective of our investigations is to eliminate "guess work" involved in effective use of parallel, multi-resolution simulations.

References

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

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

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

[4] S. Prakash and R. Bagrodia. MPI-SIM: Using parallel simulation to evaluate MPI programs. In *Winter Simulation Conference*, pages 467–474, 1998.

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

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

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

[8] D. M. Rao and P. A. Wilsey. Performance prediction of dynamic component substitutions. In *In Proceedings of the 2002 Winter Simulation Conference (WSC'02)*, Dec. 2022.

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

[10] A. F. Sisti and S. D. Farr. Model abstraction techniques: An intuitive overview. *Publication of the AFRL/IF*, 1998.