Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

PERFORMANCE PREDICTION OF DYNAMIC COMPONENT SUBSTITUTIONS

Dhananjai M. Rao
Philip A. Wilsey

Experimental Computing Laboratory
Dept. of ECECS, PO Box 210030,
Cincinnati, OH 45221–0030. U.S.A.

ABSTRACT

The Web-based Environment for Systems Engineering (wese) is a web-based modeling and simulation environment in which the level of abstraction of a model can be configured *statically* (prior to simulation) or *dynamically* (during simulation) by substituting a *module* (set of components) with an equivalent component or vice versa through a process called Dynamic Component Substitution (DCS). DCS can considerably improve the overall efficiency of simulations by enabling dynamic tradeoffs between several modeling and simulation related parameters. However, identifying ideal sequence of DCS is a complicated task. This paper proposes a novel methodology called *DCS performance prediction methodology* (DCSPPM) to identify ideal sequences of DCS. DCSPPM utilizes estimates of the changes induced by each *atomic* DCS along with model characteristics to predict the changes induced by a combination of substitutions. Our studies indicate that the proposed methodology provides good estimates (maximum error < 8%) of the changes induced by DCS.

1 INTRODUCTION

Web-based simulations are steadily growing in importance because they are an effective solution to address several issues exacerbating modeling, simulation, and analysis of modern systems (Rao and Wilsey 2000). To ease web-based modeling and distributed simulation, a Web-based Environment for System Engineering (wese) has been developed (Rao and Wilsey 2000). wese provides a hierarchical, component-based modeling language called the System Specification Language (SSL). In SSL, a system is represented as a set of interconnected components. A component is a well defined *atomic* entity which is viewed as a "black box" — *i.e.,* only its interface and functionality is of interest and not its implementation. A set of components that model a sub-system can be grouped into a *module*. Modules are the hierarchical building blocks of SSL. They can be reused (through a well-defined interface) in a hierarchical fashion to develop larger systems. Modules can be viewed as components at a higher level of abstraction. In addition to wese, component-based modeling techniques are also used in other tools because they offer several advantages (Rao, Chernyakhovsky, and Wilsey 2000).

In wese, a model may be transformed to a functionally equivalent model by substituting a module (*i.e.,* a set of components) with a *equivalent component* or vice versa. The equivalent component of a module must satisfy the following criteria: (i) it must have an interface that is identical to that of the module, and (ii) its functionality must be similar to that of the module. In this work, we do not deal with issues of establishing equivalence of a module with a component. Instead we leave the decision of equivalence to an oracle which may or may not be the modeler. In other words, when an equivalent component is specified for a module, wese assumes that it satisfies the necessary criteria. Substituting a module with its equivalent component or vice versa is synonymous to varying the level of abstraction and the resolution of the model (Rao and Wilsey 2000). Figure 1 shows different transformations that can be applied to a typical full adder (digital logic). For example, Figure 1(b) illustrates the modules `exclusive-or gate` and `2-bit mux` (shown in Figure 1(a)) substituted with equivalent components. Transformations to a model can be performed *statically* or *dynamically*. Static transformations occur prior to simulation while dynamic transformations occur during simulation.

In wese, static and dynamic transformations to a model are effected through a process called Dynamic Component Substitution (DCS) (Rao and Wilsey 2000). DCS can be used to enable optimal, dynamic tradeoffs between several interrelated modeling and simulation parameters such as: modeling costs, resolution of the model, accuracy of results, and simulation performance (Rao, Chernyakhovsky, and Wilsey 2000, Rao, Wilsey, and Carter 2001). It has shown to be an effective technique to improve the overall efficiency of a simulation study (Rao and Wilsey 2000, Rao,

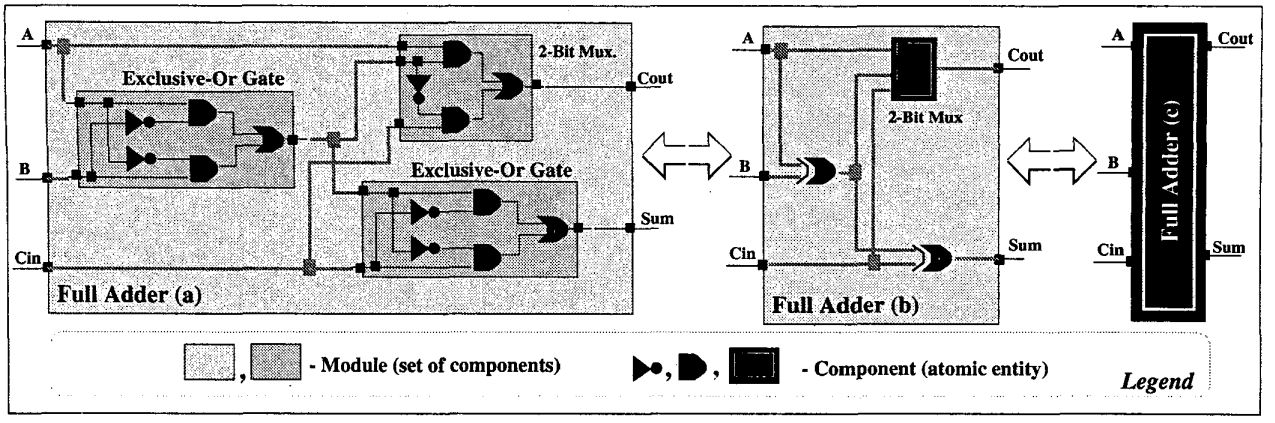


Figure 1: Functionally Equivalent Models for a Full Adder

Wilsey, and Carter 2001). For example, parts of a model that are inconsequential to a given study can be abstracted in order to improve the overall simulation time. However, the impact of a DCS is dependent on the model; *i.e.,* a given DCS may improve, deteriorate, or have no impact on the simulation performance. Therefore, it is crucial to identify and utilize ideal combinations (or sequences) of DCS in order to improve the overall efficiency of a simulation study. However, to determine an ideal sequence of transformations, exhaustive analysis of the possible combinations of transforms must be performed. The analysis is further complicated in the case of web-based simulations because they involve several asynchronous, concurrent operations. Exhaustively analyzing combinations of transformations results in combinatorial explosion of the problem space and is not a realistic approach even for medium sized models.

In an endeavor to engineer a more practical approach for identifying and utilizing efficiency improving substitutions, this paper proposes a novel methodology called *DCS prediction methodology* (DCSPPM). DCSPPM identifies efficiency improving DCS transformations using quantitative measures for generated through a combination of: static analysis of the model, empirical measures of the event granularities (time taken to process an event) of components, estimates of the communication latencies between workstations used for parallel simulation, and by applying heuristics to predict synchronization overheads. This paper presents the design, implementation, and testing of DCSPPM in wese. Section 2 presents an overview of wese. Section 3 presents a detailed description of DCSPPMalong with the issues involved in the implementing DCSPPM in wese. Some of the experiments conducted to evaluate the accuracy of the estimates generated by DCSPPM are discussed in Section 4. Section 5 concludes the paper and presents some of the ongoing work.

2 WESE

This section presents only a brief overview of weseto aid further discussions in the remainder of the paper. A detailed description of wese and DCS is available in the literature (Rao, Chernyakhovsky, and Wilsey 2000, Rao and Wilsey 2000, Rao, Wilsey, and Carter 2001). wese provides a component based modeling language, a framework for developing a web-based repository of components, and the infrastructure for distributed simulation. An overview of wese is shown in Figure 2. wese provides both an HTML interface and a text based frontend that can be used to interact with the wese server. The server controls and coordinates the various parallel and distributed activities of the system. The primary input to wese is the model of the system described using the System Specification Language (SSL). The specification of a model or an SSL design file consists of a set of interconnected *modules*. Each module consists of three main sections, namely: (i) the *component definition section* that contains the details of the components to be used to specify a module (such as the Universal Resource Locator (URL) of a factory and name of the source object along with initial parameters); (ii) the *component instantiation section* that defines the various components constituting the module; and (iii) the *netlist section* that defines the interconnectivity between the various instantiated components. SSL permits an equivalent component 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* to be associated with each module. The *label* can be used as a 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 2, the input SSL source is parsed into an

object-oriented (OO) in-memory *intermediate form* (SSL-IF). Hierarchical SSL models are elaborated or "flattened" prior to simulation by the elaborator (Rao, Chernyakhovsky, and Wilsey 2000). Elaboration is a recursive process that flattens a hierarchical model by substituting each module reference (made through the use of *labels*) with a unique instance of the module.

The `wese` server also performs the task of collaborating with the distributed factories and coordinating the simulations. The DCSPPM module houses the implementation of the proposed DCSPPM. A detailed description of DCSPPM is presented in Section 3. The *simulation manager* (Figure 2) performs the activities associated with coordinating with the object factories (via the *factory manager*) to setup a distributed simulation. 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 infrastructure (Rao, Chernyakhovsky, and Wilsey 2000). Parallel simulations are performed by utilizing components (or simulation objects) from different factories. a `wese` factory is built from sub-factories and *object stubs*. Object stubs contain attributes of the a component such as interface description, cost, and formal specifications. The *simulation sub-system* of a `wese` factory is built around the `warped` simulation kernel. `warped` is an API for a general purpose discrete event simulation kernel with different implementations (Radhakrishnan et al. 1998). `wese` utilizes the Time Warp (Radhakrishnan et al. 1998) based simulation kernel of `warped`. It provides the infrastructure for distributed simulation and also performs the task of enabling DCS. A more detailed description of `warped` and Time Warp are available in the literature (Jefferson 1985, Radhakrishnan et al. 1998).

In `wese`, an event-driven mechanism has been employed to sequence the various phases involved in DCS. A component can trigger DCS by merely scheduling an appropriate kernel event. `wese` also provides a simple API for mapping states of components during DCS.

3 DCS PERFORMANCE PREDICTION METHODOLOGY

The DCSPPM has been developed to ease exploration of different configurations of a model to determine sequences of efficiency improving DCS transformations. Prior to DCSPPM, several techniques were explored to predict the changes in performance induced by DCS (Rao, Wilsey, and Carter 2001). However, the results obtained from these techniques had considerable errors (in the range of $\pm 30\%$ to $\pm 50\%$), particularly in parallel simulation scenarios. Since the error factors were large, the earlier techniques were practically unusable. Consequently, one of the primary motivations for developing DCSPPM was to design a more accurate methodology.

In DCSPPM, identification of DCS transformations is performed by comparing the empirical estimates generated by DCSPPM for each DCS transformation. The empirical estimates generated by DCSPPM indicate the *changes* induced by a transformation on various model and simulation related parameters such as: modeling costs, observability of the model, and change in simulation performance. The estimates can also be viewed as weights associated with each transformation. Consequently, identifying ideal sequences of transformations can be reduced to an optimization problem of choosing a sequence of transformations such that the sum of their weights is optimal.

For example, consider a scenario that involves three DCS transformations, say t_1, t_2, and t_3, and DCSPPM is used to estimate the changes induced by these transformations. An example of the change in observability generated by DCSPPM would be -10%, -5%, -5% for t_1, t_2, and t_3, respectively. Let the change in simulation times estimated by DCSPPM be $+10\%$, -5%, and -5%. Positive values indicate increase in the quantitative estimate of the given parameter while negative values indicate decrease. Let us assume that the objective is to minimize simulation time with minimal decrease in observability. In this case, an ideal sequence of DCS can be chosen by selecting those transformations which decrease in simulation time is better than the decrease in observability *i.e.,* t_2 and t_3 are the candidates while t_1 is not. In other words, a solution to the given problem would be to use transformations t_2 and t_3 and ignore transformation t_1.

It must be noted that the modeling and simulation costs are independent of each other. However, in practice they must be simultaneously optimized in order to enable ideal tradeoffs. The quantitative estimates generated by DCSPPM are a measure of the *changes* induced by a transform and are not absolute measure. In other words, the goal is to identify the best combination given a set of choices and not the absolute optimal configuration for a given model. In DCSPPM, the changes induced by a transformation on the various parameters are estimated through a combination of: static analysis of the model, empirical measures of the event granularities (time taken to process an event) of components, estimates of the communication latencies between workstations used for parallel simulation, and by applying heuristics to predict synchronization overheads. A detailed description of the techniques used by DCSPPM to generate the quantitative estimate is presented in the following sub-sections.

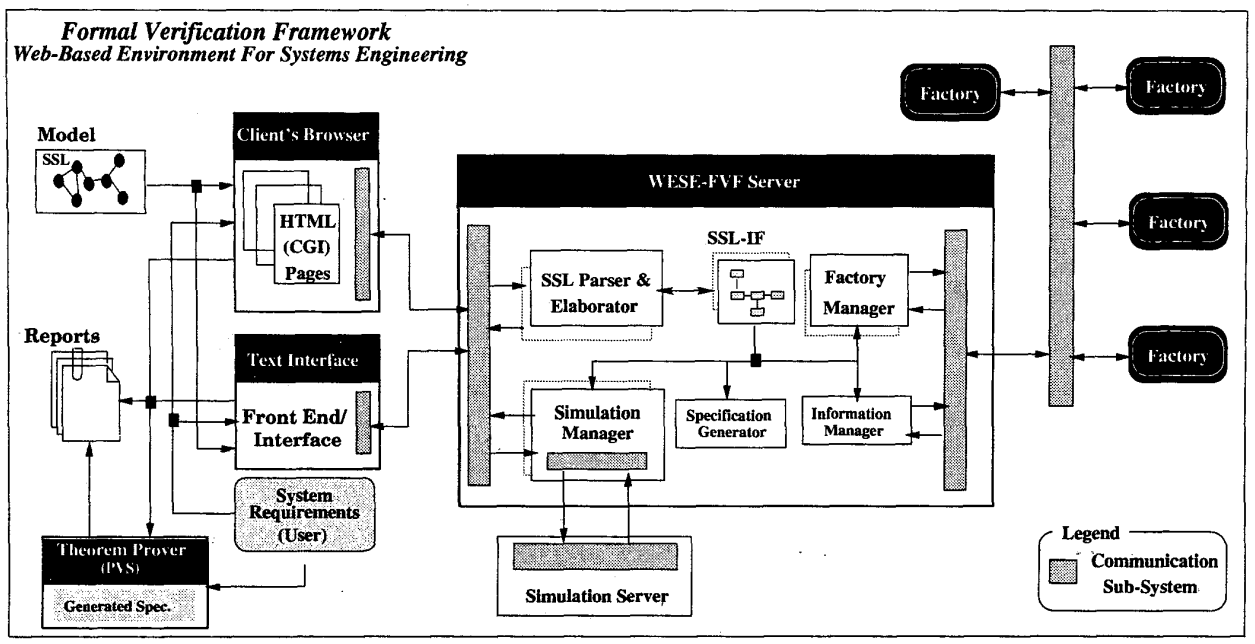


Figure 2: Overview of wese

3.1 Estimation of changes in Modeling Parameters

The model related quantitative estimates generated by DCSPPM are: change in cost of model, changes in observability, and change in level of abstraction. These quantitative estimates are dependent solely on the components constituting the model. The changes in modeling costs is an abstract quantity that characterizes the overheads involved in developing or using the components. It is an important measure, particularly in web-based simulations because components obtained by third-party model developers may be used. The third party components may are typically offered as value added services based on different pricing schemes (Rao, Wilsey, and Carter 2001). In DCSPPM the cost of a model is computed as the sum of the costs of the components constituting the model. Different schemes may be used to determine the cost of a component. In wese, the number of lines of source code for each component have been used as a measure of cost. The cost of each component is available as a part of the *object stub* associated with each component in a wese factory (please refer Section 2). The cost of the components is collated by the DCS module present in the wese server (Figure 2). The changes induced by a transformation to the modeling costs is computed as the percentage change in the overall cost of the model when a module is substituted by its equivalent component (or vice versa).

The change in the total number of ports in the model is used as a measure of the change in observability. In wese, a *port* is a conceptual point through which some interaction occurs with a component and a set of ports constitutes the interface of a component. The number of ports of a component are a part of the SSL description of the model. The total number of ports are computed by summing up the number of ports of each component in the model. When a DCS transformation is applied to a model, the components constituting the model changes. Correspondingly, the total number of ports in the model also change. The percentage change in the total number of ports induced by a transform, is reported as a measure of the change in observability. In DCSPPM the change in the level of abstraction is estimated by computing the percentage change in the total number of components and hierarchical levels. The change in observability and level of abstraction is computed by statically analyzing the elaborated SSL description of the model. The object-oriented nature of SSL-IF has been utilized to implement the static analyses. The DCSPPM module, present in the wese server (Figure 2) handles the task of generating the estimates using the above methodology. The time complexity of this phase of DCSPPM is $O(c)$, where c is the total number of components in the model.

3.2 Estimation of changes in Simulation Performance

The simulation parameters are dependent on the model as well as the hardware platform used for simulation. The primary parameter computed by DCSPPM is the overall

change in simulation time when a transformation is applied to a model. The change in simulation time is measured in terms of the change in the granularity of the model. The granularity of a model is in turn determined by the granularity of the components constituting the model, the platform used for simulation, and the configuration of the simulation (such as number of processors used and partitioning of components).

The granularity of a component represents the average time taken by the component to process an event. In DCSPPM, three factors contribute to the granularity of a component; namely (i) average time taken to process an event; (ii) communication costs involved in receiving the event over communication networks (if any); and (iii) synchronization overheads. These parameters have shown to determine th e overall time taken to simulate a model (Balakrishnan et al. 1997). The techniques used to estimate these three parameters are discussed below.

3.2.1 Event Processing Cost

The event processing cost represents the average time taken to execute one event. The cost of processing an event also includes the simulation kernel overheads such as state saving overheads and event scheduling costs. The event processing costs of a component are experimentally determined by setting-up a temporary "test" simulation and monitoring the time taken to execute each event. The granularity estimation is performed by the `wese` factory which houses the component. It must be noted that the simulation is also performed by the same factory (or workstation). The `wese` factory provides an API that must be used by the component-developer to define the test simulation to be used. The granularities are assumed to follow a Normal distribution in concordance with statistical theories (Hogg and Craig 1995, Jain 1991). Suitable (95%) confidence intervals are also computed and stored in the stubs.

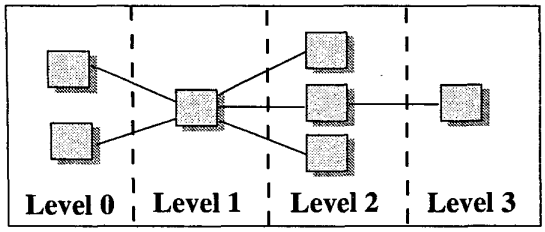
The API also provides support for estimating the event processing costs for components that have multiple, distinct regions — the time taken to process an event significantly varies (based on the modeler's discretion) from event to event. In this case, each distinct region is assumed to follow a Normal distribution (as before) and the overall event processing is defined as a weighted average of each individual distribution. The weights may also be replaced with suitable probability values which indicate the probability with which a given type of event may be received by the component. The resulting weighted average also follows a Normal distribution with a given mean and variance. In `wese`, granularities of each unique components is computed once and reused. The worst case time complexity of this phase in DCSPPM is $O(c)$.

3.2.2 Estimation of Communication Costs

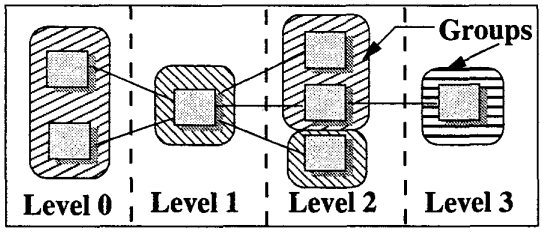
Communication latencies strongly influence the overall time taken for parallel simulations (Balakrishnan et al. 1997). In `wese`, communication latencies arise when components from two distinct `wese` factories are used to develop a module; *i.e.,* the events generated by the components have to be delivered to the target component via communication networks. On the other hand, event exchanges between components on the same factory is performed through simple pointer manipulation (by the `warped` kernel) and the overheads are included as a part of the the event processing costs (as explained above).

In DCSPPM, the communication latencies between components is estimated using the following 3 steps:

1. Levelization: During the first phase of analysis, the components and modules constituting a model are "levelized" (or ordered) such that the inputs of a component are at a lower level. Figure 3(a) illustrates an example of a levelized model. Levelization captures the flow of "inputs to outputs" in the model (from left to right in Figure 3). In other words, events in the model flow from a lower level to a higher level. The levels represent inherently serial blocks of computations in the model. Any parallelism in the model occurs in between components in each level. Since levelization requires each interconnection to be inspected, the time complexity of this phase is $O(n)$, where n is the total number of netlists (or interconnections) in the model.
2. Grouping: Next, the components at each level are grouped together based on their source factories, as shown in Figure 3(b). That is, all the components in a group reside on a given factory. The groups in each level represent the parallel entities. Note that, processing of events within a group proceeds sequentially (based on the construction of the simulation infrastructure of `wese`). The time complexity of the grouping step is $O(c)$.
3. Estimation: During the last phase, the average communication latencies between *groups* of components is estimated. Communication delays arise when events are exchanged between the *groups*; within a group the communication costs are zero (as explained earlier). Grouping of components based on factories eases identifying pairs of between which communication latencies need to be measured. Estimation of communication latencies is performed by the `wese` factories and is coordinated by the `wese` server. Latencies are estimated by exchanging a number of messages between the two factories, measuring the round trip time for the



(a) Levels



(b) Groups

Figure 3: An Example of Levels and Groups

messages, and computing an average. One wese factory acts as a server while the other acts as a client. Estimation proceeds in a "lazy" manner; *i.e.,* estimation of latencies between a given pair of wese factories is performed only once. The worst case time complexity of the estimation phase is $O(c/2)$.

The estimated average communication delays is then added to the overall granularity of the component (*i.e.,* total granularity = event processing costs + communication latency). If a component has multiple sources, then the average of the communication delay from each source is used. It must be noted that the average communication delay is also assumed to follow a Normal distribution with a given mean and variance.

3.2.3 Estimation of Synchronization Overheads

wese is a Time Warp synchronized parallel simulation environment. The *rollbacks* that occur in a Time Warp simulation are a direct measure of the synchronization overheads. In DCSPPM the synchronization overheads are represented as the probability with which a component would be rolled-back during simulation. In a Time Warp simulation, rollbacks imply that some of the events must be reprocessed. Accordingly, the overall granularity of the component is increased by this probability factor to account for synchronization costs —*i.e.,* the average granularity of the component increases (by a given percentage) because of rollbacks ($total\_granularity = total\_granularity + (total\_granularity * rollback\_probability)$).

DCSPPM uses a heuristic to estimate the probability of a rollback. The intuition behind this heuristic is that a component will be rolled-back if, concurrent events (events with same simulation time) arrive at different times (real time). The probability of such occurrences increases as the variance in event arrival times at the inputs of component increases. For example, if all the inputs were being generated by components on the same factory, then the probability of a rollback is almost zero. On the other hand, if the inputs were being generated by components with different total granularities, on different factories, the probability of a rollback increases.

The levelized and grouped model (generated earlier) is also used to estimate synchronization costs. The estimation proceeds from the lowest layer to the highest layer, tracing the "natural" flow of events in the model. At each level, the synchronization costs for each component is computed using the proposed heuristic and the total granularity of the components from earlier levels. The rollback probability of components at a lower level is also taken into consideration in order to account for cascading rollbacks. The results from the static analyses are stored back into the intermediate form for future references. The time complexity of this phase of DCSPPM is $O(c * n)$.

3.3 Identifying Efficiency Improving Sequences of DCS

Having estimated the cost, observability, and average granularity of each component, the overall cost, observability, and average granularity of the complement model is computed. This performed my merely summing up the attributes of each component in the model. Changes induced in these attributes by a DCS transformation (*i.e.,* when a module is replaced by a component) is also computed. The change in attribute value is computed as follows. Let a given DCS transformation substitute a module m containing the set of components $m = \{c_1, c_2, \cdots, c_n\}$ by $c_{EC} (c_{EC} \notin m)$. Then, the change in a given attribute a, represented by $\Delta(a)$, is computed as:

$$\Delta(a) = (\sum_{\forall c_i \in m} c_i.a) - c_{EC}.a$$

where $c_x.a$ represents the quantitative estimate of attribute a for component c_x. Moreover, each DCS transformation also involves additional overheads during simulation. These overheads are estimated in terms of the number of kernel events generated to achieve DCS. In wese, three kernel events (two `update` events, and one `state-value` event) are scheduled for each port in the module being substituted. An average granularity for each kernel event is estimated by each wese factory and that average is multiplied three times by the number of ports in the module to obtain the DCS overheads — *i.e.,* $DCSoverhead = 3 * |ports| * (average\ DCS\ cost\ of\ one\ port)$. The number of ports in a module is obtained from SSL-IF. All arithmetic operations

are performed using statistical operations defined for Normal distributions (Hogg and Craig 1995). The worst case time complexity of this phase of the algorithm is $O(c)$.

DCSPPM utilizes the above described techniques to generate quantitative estimates of the changes induced by a DCS transformation in modeling costs, observability of the model, and simulation performance. These estimates are then used to identify an ideal sequence of DCS based on the user's requirements (as explained in Section 3). The modeler may also manually choose the sequence of DCS based on the estimates generated by DCSPPM. The overall time complexity of DCSPPM is $O(3.5c + n + cn)$, where c is the total number of components and n is the number of interconnections (or netlist entries) in the model. In general, assuming $c << n$, $O(DCSppm) \approx O(n)$.

3.4 Assumptions underlying DCSPPM

DCSPPM is a static parameter estimation methodology. Several assumptions regarding the model characteristics and the simulation platform have been made during its design and implementation. The assumptions underlying DCSPPM are: (i) the underlying simulation kernel scales linearly with respect to the number of events; (ii) the overheads of enabling DCS is linear with respect to the number of ports in a module; (iii) workload on the workstations does not significantly change during simulation; (iv) communication latencies do not change considerably during simulation; (v) overall granularity of the models does not skew considerably; *i.e.,* the probability with which a component may receive events with different granularities is the same; and (vi) if the model has several different paths from inputs to outputs, then the probability with which each path is taken is equal. The last two assumptions imply that DCSPPM assumes that the behavior of the model (in a given simulation-run) does not deviate significantly from its average behavior. If the behavior is skewed then, in such scenarios the estimates generated by DCSPPM will be inapplicable.

4 EXPERIMENTS

The experiments conducted to evaluate the accuracy of the estimates generated by DCSPPM are presented in this section. The experiments were conducted using a set of digital logic circuits (real world models) and a set of synthetic models. The synthetic models were used to obtain larger benchmarks with a broader range of characteristics and behaviors. They were developed suitably re-targeting the Performance and Scalability Analysis Framework (PSAF) (Balakrishnan et al. 1997) backend. PSAF provides a platform-independent Workload Specification Language (WSL) that allows characterization of simulation models using a set of fundamental performance critical parameters. A `wese`-specific backend was developed for PSAF in order to obtain the synthetic models. A more detailed description of PSAF along with the API for developing new PSAF-backends is available in the literature (Balakrishnan et al. 1997).

Some of the characteristics of the benchmarks used in the experiments are shown in Table 1. These benchmarks were described in SSL by suitably utilizing components from various `wese` factories. Larger models were built from smaller sub-modules using the hierarchical model technique supported by SSL. The models also included equivalent component specifications for modules, that get used during DCS. For example, the `4-Bit-Adder` (shown in Table 1) is implemented using a set of `Full Adders`. Each `Full Adder` is specified using a set of basic gates along with auxiliary component specifications (Figure 1). Parallel simulation experiments were conducted by suitably modifying the SSL descriptions to utilize components from a given number of `wese` factories. For parallel simulation, the `wese` factories are deployed on a network of shared memory multiprocessor (SMP) workstations running Linux. Each workstation consists of two 166MHz Pentium Pro Processors with 128MB of memory. Two factories are deployed per workstation and the workstations are networked using fast Ethernet.

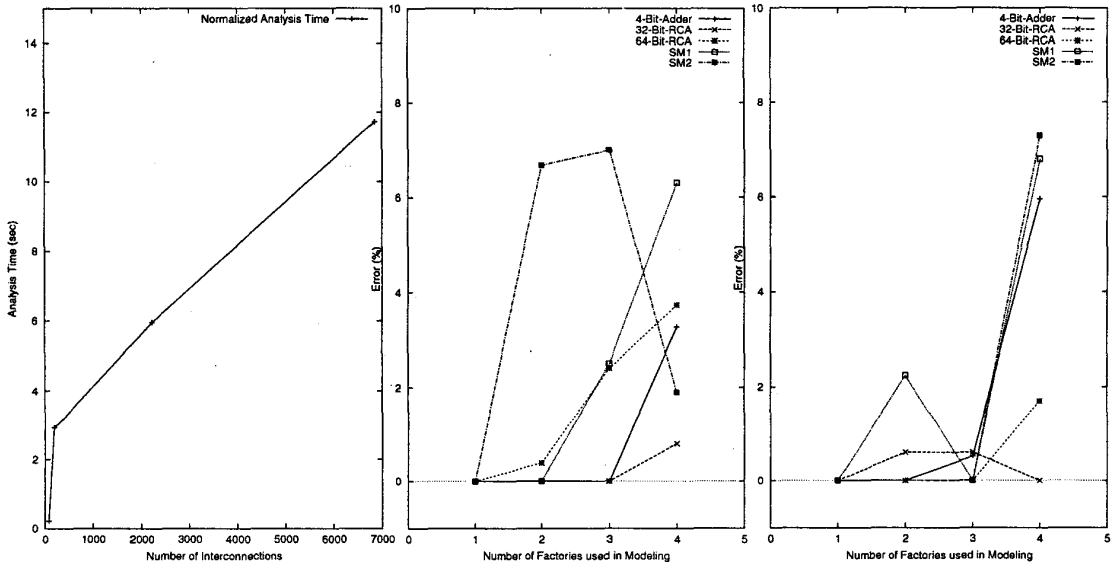
The time taken for analyzing different configurations of the models, using a varying number of `wese` factories is shown in Table 1. The number of modules (shown in Table 1) in each benchmark also indicates the number of DCS transformations that had to be analyzed. The timing information shown is the graph is the average of 10 runs. The analysis times shown in Table 1 also include the time taken to estimate the communication latencies between different `wese` factories. Figure 4(a) illustrates the average analysis time without communication delays for the different model configurations. The timing in Figure 4(a) has been normalized with respect to the number of interconnections (or netlists) present the models. As shown in Figure 4(a), the time for analyzing a model varies linearly with respect to the number of interconnections (or edges) in the model. The graph confirms the expected time complexity (Section 3) of DCSPPM to be approximately $O(n)$, where n is the number of interconnections in the model.

The graphs in Figure 4(b) and Figure 4(c) presents the error in the estimates of simulation time generated by DCSPPM for both static and dynamic component substitution cases. The error percentages were computed by comparing the predicted changes in simulation time against the observed changes. The error value indicates the deviation of the observed data from the 95% confidence interval of the corresponding value predicted by DCSPPM. The simulations involving dynamic transformations did not involve any static transformations (and vice versa) in order to clearly distinguish the results obtained in the two cases. In addition, no additional jobs were run on the various workstations used for simulation — *i.e.,* the load on the workstations was

Table 1: Details of Modules used for Experiments

Model Name	Total Number of				Total DCS Analysis Time (sec)			
	Hierarchies	Modules	Components	Netlists	1 F^o	2 F^o	3 F^o	4 F^o
4-Bit-Adder	3	4	30	82	0.73	2.51	5.16	6.48
32-Bit-RCA*	4	64	192	1088	9.10	10.30	12.32	13.25
64-Bit-RCA*	5	128	884	2224	17.82	18.42	19.82	20.36
SM^+1	4	200	2000	4375	31.06	30.90	31.47	31.48
SM^+1	4	300	3000	6850	46.73	45.62	45.19	44.58

Note: *RCA = Ripple Carry Adder; $^+$SM = Synthetic Model; ^{o}F = Factory



(a) Normalized analysis Time (b) DCSPPM Error (Static) (c) DCSPPM Error (Dynamic)

Figure 4: Results from Empirical Evaluation of DCSPPM

almost a constant throughout the experiments. It must be noted that the errors in estimates of model related parameters (such as change in costs and observability) were zero because they are deterministic estimates. In other words, they are generated through static analysis of the model and not using empirical estimates.

As illustrated by the graphs in Figure 4(b) and Figure 4(c), the estimated change in simulation time closely reflects the observed change in simulation time. The maximum error in the estimations was about 8% even though the predicted variance in the estimated values are relatively small ($\pm 2\%$ to $\pm 7\%$). As illustrated by the experiments, the predicted changes in simulation time (with 95% confidence intervals) closely track the observed changes in simulation time demonstrating the accuracy of the estimation technique used in DCSPPM. The estimates in the case of 1 factory simulations (inherently sequential) are accurate because of the absence of non-deterministic factors such as communication latencies and rollbacks. As illustrated by the graphs in Figure 4, the estimates generated by DCSPPM closely track the actual changes that occur during simulation. The experiments highlighting the effectiveness of the estimation methodology used in `wese`.

5 CONCLUSIONS

The design and implementation of DCSPPM, a methodology for performance estimation of static and dynamic component substitution, was described in this paper. DCS, coupled with DCSPPM, is an effective technique to enable more optimal tradeoffs between several model and simulation related parameters. They make `wese` a controlled environment for conducting simulations — a model developer can utilize the estimates to intelligently fine tune the simulations to

achieve maximum efficiency. DCSPPM has a polynomial time complexity and significantly reduces the overheads involved in exhaustively analyzing all possible combinations of DCS transformations. For example, a straightforward "greedy" algorithm (similar to 0/1 Knapsack algorithm) can be employed to obtain a sequence of transforms that optimize a model for a given combination of the modeling and simulation parameters (an optimizing function along one or more axes). It must be noted that DCSPPM aims to identify the best combination given a set of choices and not the absolute optimal configuration for a given model. The experiments presented in this paper show that the predicted changes closely track (with an error of $\pm 8\%$) the observed changes, highlighting the effectiveness of DCSPPM. The estimates may also be used as indicators for further model development and refinement. Currently, work is underway to relax some of the assumptions underlying DCSPPM. Studies are also being conducted to adapt DCSPPM for conservatively synchronized parallel simulations. As indicated by our studies, DCSPPMprovides a effective methodology to estimate the changes induced by a sequence of DCS in several modeling and simulation parameters.

ACKNOWLEDGMENTS

Support for this work was provided in part by the Ohio Board of Regents.

REFERENCES

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*.

Hogg, R. V., and A. T. Craig. 1995. *Introduction to mathematical statistics*. Englewood Cliffs, New Jersey: Prentice Hall.

Jain, R. 1991. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. New York, NY: Wiley-Interscience.

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

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

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

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

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.

AUTHOR BIOGRAPHIES

DHANANJAI M. RAO <dmadhava@ececs.uc.edu> is a Ph.D. candidate in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received his Masters from the same department in August 2000. He received his Bachelor's degree in Computer Science and Engineering from the University of Madras, India in 1996. His research interests include parallel discrete event driven simulation, distributed computing, network simulation, object oriented design patterns, and web-based simulation.

PHILIP A. WILSEY <philip.wilsey@uc.edu> is an Associate Professor in the Department of Electrical & Computer Engineering and Computer Science at the University of Cincinnati. He received PhD and MS degrees in Computer Science from the University of Louisiana at Lafayette and a BS degree in Mathematics from Illinois State University. His current research interests are parallel and distributed processing, parallel discrete event driven simulation, computer aided design, formal methods and design verification, and computer architecture. He is a senior member of the IEEE and is a member of the IEEE Computer Society and the ACM.