VLSI DESIGN
1999, Vol. 9, No. 3, pp. 237–251
Reprints available directly from the publisher
Photocopying permitted by license only

Dynamic Cancellation: Selecting Time Warp Cancellation Strategies at Runtime*

RAGHUNANDAN RAJAN, RADHARAMANAN RADHAKRISHNAN and PHILIP A. WILSEY†

Computer Architecture Design Laboratory, Dept. of ECECS, PO Box 210030, Cincinnati, OH 45221-0030

(Received 26 May 1998)

The performance of Time Warp parallel discrete event simulators can be affected by the cancellation strategy used to send anti-messages. Under aggressive cancellation, anti-message generation occurs immediately after a straggler message is detected. In contrast, lazy cancellation delays the sending of anti-messages until forward processing from a straggler message confirms that the premature computation did indeed generate an incorrect message. Previous studies have shown that neither approach is clearly superior to the other in all cases (even within the same application domain). Furthermore, no strategy exists to make *a priori* determination of the more favorable cancellation strategy. Most existing Time Warp systems merely provide a switch for the user to select the cancellation strategy employed. This paper explores the use of simulation time decision procedures to select cancellation strategies. The approach is termed *Dynamic Cancellation* and it assigns the capability for selecting cancellation strategies to the Logical Processes (LPs) in a Time Warp simulation. Thus, within a single parallel simulation both strategies may be employed by distinct LPs and even across the simulation lifetime of an LP. Empirical analysis using several control strategies show that dynamic cancellation always performs with the best static strategy and, in some cases, dynamic cancellation provides some nominal (5–10%) performance gain over the best static strategy.

Keywords: Discrete-event simulation, feedback control, dynamic adjustment, parallel simulation, Time Warp

1. INTRODUCTION

The Time Warp mechanism is one of the most important synchronization protocols for parallel simulation [1]. However for most applications, the successful use of Time Warp requires the careful selection of Time Warp optimization parameters (*e.g.*, cacellation strategies, frequency of state saving, and so on). Furthermore, the optimal setting for the simulation parameters may not hold

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

† Corresponding author. Tel.: (513) 556-4779; e-mail: phil.wilsey@uc.edu

across an application domain or even throughout the entire simulation lifetime of a single application. Consequently, several investigations have proposed the dynamic adjustment of simulation parameters over the lifetime of the simulation [2–8]. These investigations have shown that dynamic parameter adjustment can be used to successfully tune a Time Warp simulator for improved performance.

The dynamic adjustment of simulation parameters requires careful design considerations. Many of these considerations are quite similar to the problems studied by traditional linear and adaptive control theorists [9]. These considerations include the need to implement an adjustment mechanism that converges to stable values and the need to understand how parameter adjustment affects (sampled) output values [10]. However, dynamic adjustment of simulation parameters occurs by using general CPU cycles and thus introduces overhead into the simulation. This overhead must be carefully controlled – too little processing and the control system may not respond correctly; too much processing and the overall simulation performance may deteriorate. Thus, Time Warp control systems should be employed only as a last resort, when traditional static analysis fails.

One important tradeoff in a Time Warp simulation is which cancellation strategy to employ: *aggressive cancellation* [11] or *lazy can cellation* [12, 13]. Numerous studies of cancellation strategies have been performed and none has yet been able to clearly demonstrate when one strategy consistently outperforms the other [2, 14]. The studies have only succeeded in showing that aggressive cancellation is more sensitive to optimal parameter adjustments in the lower level communication subsystem. Thus, most currently available Time Warp simulators support both strategies and provide user control of their setting [15–17]. Furthermore, while Lin [18] has attempted to analytically characterize the effective performance spaces of each cancellation strategy, the approach requires complete knowledge of simulation behaviors and is not practically applicable. Thus, the determination of cancellation strategy would seem to be a suitable candidate for dynamic selection. Its candidacy is further supported if one considers that even within a simulation, different processing sub-assemblies may operate more efficiently with opposite cancellation strategies.

This paper discusses the construction of control systems to dynamically select cancellation strategies. More precisely, this research proposes that Time Warp simulations should be constructed such that each Logical Process (LP) maintains (by dynamic adjustment) a binary value denoting which cancellation strategy to use on rollback. This dynamic selection is termed *Dynamic Cancellation* and its utility is evaluated in this paper. The investigation in support and evaluation of dynamic cancellation is organized into three parts: (i) instrumentation and analysis of an existing Time Warp implementation to evaluate the potential utility of dynamic cancellation (do different LPs favor different cancellation strategies? does an LP's processing behaviors vary over its lifetime such that the behavior favors varying the cancellation strategy?); (ii) the design and implementation of control mechanisms for selecting a cancellation strategy; and (iii) an analysis of the performance of dynamic cancellation in comparison to either static (lazy or aggressive) strategy.

2. MOTIVATION FOR DYNAMIC CANCELLATION

The Time Warp synchronization strategy for parallel discrete event simulation [11] offers several attractive properties over conservative approaches [19] and has the potential to outperform them [20]. This potential arises due to Time Warp's relaxed enforcement of causality constraints. Unfortunately, instances of Time Warp simulators have yet to reliably produce improved performance figures. This failure is caused by the fact that Time Warp simulators are sensitive to internal parameter settings and when set incorrectly, Time

Warp suffers from several problems such as instability (due to excessive rollbacks), wasted premature computation, and excessive memory usage [10]. The amount of wasted computation can be reduced by minimizing the number of rollbacks in the system. A variety of algorithms have been proposed to alleviate the problem of excessive memory usage due to frequent state savings [5, 21–23]. All of these algorithms use simulation time information, such as the number of rollbacks, to suggest an optimal checkpoint period. These studies show that overall Time Warp performance can be improved by dynamic parameter adjustment.

In conventional Time Warp, two strategies exist to undo the effects of the erroneous computation, namely aggressive [11] and lazy cancellation [12, 13]. Under aggressive cancellation the arrival of a straggler message causes the immediate generation of anti-messages for all output messages that were prematurely dispatched. In contrast, under lazy cancellation the sending of anti-messages is delayed until forward processing demonstrates (by comparison of old and new output) that the originally sent output messages were incorrect. Lazy cancellation has the potential to reduce the communication overhead as well as the wasted premature computation, provided the probability of the regenerated messages being the same as the prematurely sent output messages, is high. Performance under lazy cancellation deteriorates as the probability that the regenerated messages are different from the previously sent output messages increases. This occurs because delaying the dispatch of anti-messages prolongs the processing of incorrect events (which is magnified by the propagation of rollbacks throughout the simulation). Aggressive cancellation immediately discards all the premature computation done by an LP and tends to perform less effectively than lazy cancellation if the same messages tend to be regenerated after a rollback. The performance of a Time Warp simulator, therefore, depends on the efficiency of the cancellation mechanism that is employed to contain the spread of the erroneous computation. Several independent studies have shown that while some applications slightly favor lazy cancellation, other executions perform significantly better under aggressive cancellation (even within the same application domain) [2, 8, 24]. Unfortunately, practical techniques to statically analyze an application for selecting cancellation strategies have yet to be developed [18]. Consequently, to obtain optimal performance, some kind of mechanism must be provided to automatically select the best cancellation strategy for the specific program being simulated.

This research explores the feasibility of using control theoretic techniques to implement dynamic cancellation. We attempt to reduce the number of rollbacks by providing *metrics* that an LP can use to select the cancellation strategy that supports the most efficient generation of anti-messages. Ideally this means that the premature computation will be bounded so that no causality violations occur (and hence, no anti-messages are generated). Practically we use the term "most efficient" to indicate that

1. the earliest possible dispatch of an anti-message occurs whenever such messages are to be generated (aggressive cancellation), and
2. the non-generation of an anti-message when reprocessing after a straggler will regenerate the original message (lazy cancellation).

Throughout this paper, the term "most favorable cancellation strategy" will be used to denote the cancellation strategy that produces the most efficient generation of anti-messages. Both aggressive and lazy cancellation mechanisms as well as the new dynamic cancellation proposed herein, have been implemented as part of a Time Warp parallel discrete event simulation kernel, called WARPED [17, 25].

The remainder of the paper is organized as follows. Section 3 contains a brief overview of non-linear control theory and a description of Dynamic Cancellation. The interactions of this heuristic with the other optimizations is also presented. Section 4 describes the environment and the benchmarks that were used to evaluate the

proposed approach. This is followed by a comparison of results obtained using dynamic cancellation with those obtained using lazy and aggressive cancellation. Finally, Section 5 contains some concluding remarks and identifies areas for future work.

3. TIME WARP CANCELLATION STRATEGIES

The performance of a Time Warp simulator depends on the efficiency of the cancellation strategy employed to undo the effects of the erroneous computation. Several independent studies have shown that neither strategy is clearly superior to the other [2, 8]. Berry [26] has documented an intriguing and beneficial characteristic of lazy cancellation. She has shown that, under some circumstances, Time Warp using lazy cancellation is capable of *supercritical speedup* (*i.e.*, it can run faster than the critical path of the simulation). The reason for this supercritical speedup is that it is possible for a correct message to be sent earlier than if it were sent following the sequence of events in the causality chain. That is, an event message sent during an erroneous computation, may turn out to be valid later on in the correct computation. Since in lazy cancellation, no anti-message is sent, the other LPs are ahead relative to the progress they would have made strictly according to the directed graph from which the critical path is calculated.

Need for Dynamic Selection of Cancellation Strategies

Depending on the characteristics of a particular application and the nature of the input data, either aggressive cancellation or lazy cancellation can provide the better performance. In studies involving the simulation of digital systems [27] using the hardware description language VHDL [28, 29], we have observed that:

- Neither cancellation strategy is clearly superior.
- The optimal cancellation strategy is sensitive to the partitioning scheme employed for distributing the LPs on processors.
- Different LPs within the same application operate best under different cancellation strategies.
- Even within a single LP, the favorable cancellation strategy varies over the lifetime of the simulation.
- The optimal cancellation strategy depends on the characteristics of the application being simulated and the architecture on which the application is being simulated.

Most of the simulators that have been built using the Time Warp approach support the two cancellation strategies in the form of compile time (or simulation time) switches – leaving the responsibility of selecting the cancellation strategy to the discretion of the user. Lin's work [3] demonstrates that even with a number of unrealistic assumptions, a static analysis to determine the optimal cancellation strategy is very complicated and requires perfect knowledge of the simulation. Time Warp simulators should not expect the users to be able to adequately analyze whether their application, or a part of their application, prefers a particular cancellation strategy. In fact, the users should not even be concerned about the simulator's underlying mechanism. Thus, we propose that this selection can be performed dynamically by the simulator using a feedback control mechanism. By letting the LPs dynamically determine the cancellation strategy, an attempt is made to cancel the erroneous messages as quickly as possible without canceling messages that are likely to be correct. This results in a reduction in the amount of work that has to be undone, leading to a shorter execution time. The remainder of this section will discuss how control theory is applied to the problem of dynamic parameter adjustment.

Control theory is concerned with modifying the behavior of dynamic systems so as to achieve the

desired goals [9]. In general, a closed loop control system samples output values and adjusts input values in order to meet some performance criteria (Fig. 1). In a Time Warp system, the outputs can be a variety of things such as the ratio of lazy hits to lazy misses, the number of events committed between GVT cycles, the number of events rolled back upon receipt of a straggler message, LVT, and so on. These values are a time series of discrete values that have been used in several investigations to dynamically adjust simulation parameters [2–8].

Virtually all dynamic control investigations have also used data filtering techniques to smooth and to prevent spurious data points from causing wide variations in parameter adjustment. In this research, we have found that non-linear thresholding functions are best suited for damping the selection of which cancellation strategy to use. A thresholding function defines boundaries on input values that determine the output value produced. For example, in Figure 2, two thresholds are shown with a dead zone lying between the thresholds. In this example, the function changes its value only after it moves into the shaded region above or below each threshold. When the input falls in the dead zone, the function continues to produce the same output. Thus, the function buffers its output response to changes in the input. Since the output in our case is either aggressive or lazy cancellation, the use of thresholds is appropriate.

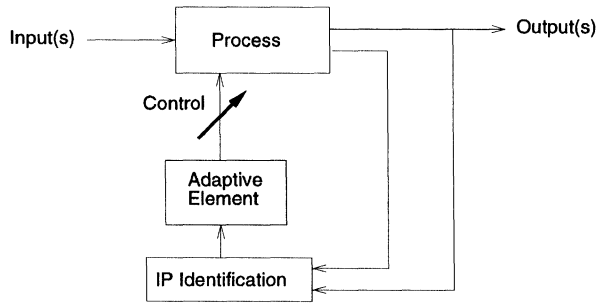


FIGURE 1 Feedback control.

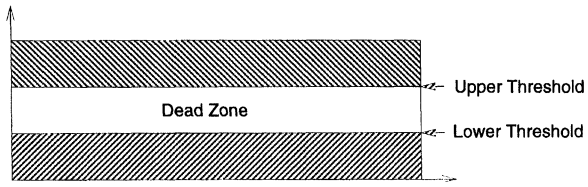


FIGURE 2 Thresholding to select cancellation strategies.

4. DYNAMIC CANCELLATION

The performance of a Time Warp simulator can be improved if the number of rollbacks can be minimized. This can be achieved by making the simulator *adapt* itself to the behavior of the application being simulated (by adapt we mean that the LPs should use the cancellation strategy most appropriate at that point in the simulation). As a result of our experiments, we have found that the cancellation strategy that a model may favor at a given point in the simulation can be predicted reasonably well by an index which we call the *Hit Ratio*. The *Hit Ratio* is a measure of how productive an LP's premature computations were in its recent past. In order to define the Hit Ratio, there are a couple of other terms that need to be defined. When using aggressive cancellation, an LP is said to have a *Lazy Aggressive Hit* if it generates the same message before and after a rollback, otherwise it is said to have a *Lazy Aggressive Miss*. These two terms have the same meaning as their counterparts used in lazy cancellation. In order to determine the cancellation strategy that the portion of the application being modeled by the LP may favor, each LP maintains a dynamic record of the past n output message comparisons. The number of comparisons, n, is statically controlled (by the user) and is called the *Filter Depth*. The *Hit Ratio* is then defined as follows:

$$\text{Hit Ratio} = \frac{\#\text{Lazy Aggressive Hits} + \#\text{Lazy hits}}{\text{Filter Depth Comparisons}}$$

The cancellation strategy is then determined using a thresholding function whose input is the *Hit Ratio*. The threshold to switch from aggressive

to lazy cancellation is called *A2L_Threshold* and the threshold to switch from lazy to aggressive cancellation is called *L2A_Threshold.* These two threshold can have the same value in which case the dead zone is eliminated. If the Hit Ratio is reasonably high *e.g.*, 0.4, then it means that the LP is favoring lazy cancellation. If, on the other hand, it is really low *e.g.*, 0.2, then it is an indication that the LP would perform well under aggressive cancellation. These thresholds are adjustable and are generally fixed at compile time. The possibility for dynamic adjustment of the threshold values exists, but is probably too expensive.

Initially, the simulation begins by using aggressive cancellation and dynamically switches between strategies based on the output of the thresholding function. That is, when a straggler arrives at an LP (and before an anti-message is dispatched) a copy of the message is placed in another queue called the *Lazy Aggressive Queue.* When the LP resumes its forward computation, every output message is compared against the messages in the Lazy Aggressive Queue, just as is done during Lazy Cancellation. If the same message is regenerated after the rollback, then it is recorded as a Lazy Aggressive Hit, otherwise it is called a Lazy Aggressive Miss. The comparisons are done in the same way as in lazy cancellation, the only difference being that here the anti-messages are sent out immediately as is done in normal aggressive cancellation. This is done to see as to what would have happened had the system been using lazy cancellation (would it have been a hit or a miss). After each comparison, the favored strategy is determined using the heuristic shown in Figure 3.

When the system is using lazy cancellation, it behaves unchanged from the normal processing of lazy cancellation. When the system is switching from lazy cancellation to aggressive cancellation, it sends out anti-messages for all the members in its Lazy Queue (the queue stores anti-messages for comparison with the regenerated messages in lazy cancellation), before switching to aggressive cancellation. When using aggressive cancellation, the simulation will aggressively send anti-messages, but it will also continue to record the result of the comparisons. In terms of space consumption, our approach has the same requirements as a system using lazy cancellation. The overhead of recording the results of the comparisons (which is done anyway when lazy cancellation is used) is nominal. The time required to switch between the two strategies is negligible compared to the time it takes to perform a rollback, unless, the number of anti-messages in the lazy cancellation queue is quite large and the LPs frequently switch between the two strategies. In this case, switching between the two strategies is more expensive than performing a rollback. Fortunately, our experiments show that the lazy cancellation queue does not generally grow very large. When a switching scheme with two thresholds is used (as in the heuristic), we have observed that the latter phenomena is infrequent (the dead zone dampens the effect of perturbations in the Hit Miss Ratio), thereby giving a reasonable speedup over systems using either lazy cancellation or aggressive cancellation.

```
if ((mode == Aggressive) and (Hit Ratio > A2L_Threshold))
    then mode := Lazy
else if ((mode == Lazy) and (Hit Ratio < L2A_Threshold))
    then mode := Aggressive
else
    Remain in the same mode
end if
```

FIGURE 3 The heuristic used for determining the cancellation strategy.

5. EXPERIMENTAL ANALYSIS

This section presents the experiments and the benchmarks used in this research along with the performance results. The first part gives a brief description of the benchmarks that were used in this investigation. The second part describes the set of experiments that were conducted to evaluate the potential utility of Dynamic Cancellation as an optimization for Time Warp. The third part discusses the performance results of the various cancellation strategies that were implemented. The experiments to evaluate the usefulness of Dynamic Cancellation were run on a 4 processor SUN SparcCenter 1000.

5.1. Benchmarks Used in this Investigation

To provide a more through analysis of the performance of Dynamic Cancellation relative to lazy and aggressive cancellation, simulation models from two different application domains were used. In particular, simulations of queuing models and simulations of digital systems described in the hardware description language VHDL [29] are used. The results presented in this paper have been obtained from two different Time Warp simulators – one of which is a purely VHDL digital system simulator, called VAST, which was developed as a part of the VAST (VHDL's Accelerated Simulation Technology) project at the University of Cincinnati, while the other is a generic Time Warp simulator called WARPED, which accepts applications written in C + +. Dynamic Cancellation has been implemented on both of the above mentioned simulators. The results for benchmarks ADDER 16 and BENCH_FPA were obtained using the VAST simulator while the results for the remaining benchmarks were obtained using the WARPED simulator. The VAST simulator uses SOLARIS threads (including the default supplied thread scheduler) and each simulation object is its own thread. The WARPED simulator organizes the simulation objects into distinct groups, each managed by an LP. The LPs execute as heavy-weight processes and simulation object scheduling is performed (by the LP) using the *Least Time-Stamp First* (LTSF) scheduling strategy.

5.1.1. Queuing Models

The following queuing models were used as benchmarks for this investigation:

RAID The RAID application models the RAID Disk Arrays which is a method of providing a vast array of storage [30] with a higher I/O performance than several large expensive disks. This application incorporates a flexible model of a RAID Disk Array and can be configured in various sizes of disk arrays and request generators. Each request is in fact a token that carries information about the number of disks, the number of cylinders, number of tracks, number of sectors, size of each sector and specific information about which stripe to read and parity information. The following configuration was used for this research: 20 source processes generate 1000 requests each to 8 disks *via* 4 forks. The whole application is partitioned into 4 LPs.

SMMP The SMMP application models a shared memory multiprocessor. Each processor is assumed to have a local cache with access to a common global memory. (The model is somewhat contrived in that requests to the memory are not serialized – *i.e.*, main memory can have multiple requests pending at any given moment). The model is generated by a C + + program which lets the user adjust the following parameters: (i) the number of processors/caches to simulate, (ii) the number of LPs to generate, (iii) the speed of cache, (iv) the speed of main memory, and (v) the cache hit ratio. The generation program partitions the model to take advantage of the fast infra-LP communication.

The numbers used for this research are: simulate 16 processors on 4LPs. The cache speed was set as 10 nanoseconds and the main memory for 100

nanoseconds. The cache hit ratio was set as 90%. This application had 100 simulation objects. Each processor generates a user specified number of memory requests. Each request (also referred to as a test vector) is in fact a token that contains information about its creation time, the creator (simulation) processor, and the time at which this request should be satisfied.

5.1.2. Digital System Simulations

The following digital circuits were used as benchmarks for this investigation. The term "simulation object" refers to gates in these benchmarks.

ISCAS2 is the c499 benchmark from the ISCAS'85 benchmark suite. It is a combinational circuit consisting of 163 simulation objects. This benchmark was tested for 2000, 5000 and 10000 input vectors.

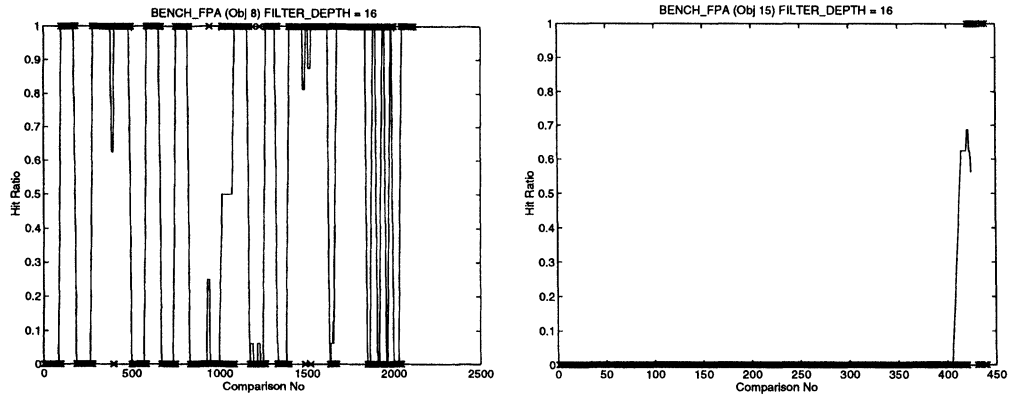
BENCH_EPA is a behavioral model of a floating point arithmetic unit and consists of 31 simulation objects. This application requires 3 test vectors, one each for the two buses in the system and another providing the control vectors for the operation of the unit. It is clock driven, therefore, the simulation proceeds as long as the clock is allowed to run. The benchmark was simulated for 3000 and 6000 femto-seconds (simulation time).

ADDER16 is a structural model of a 16-bit adder consisting of 81 simulation objects. A single test vector consisted of 2 16-bit numbers. The test vector sizes used in this application were 2000, 5000 and 10000 input vectors.

5.2. Experiments Conducted for the Investigation

As part of this investigation, we first attempted to examine trace data from a simulation for support of dynamic cancellation. In particular, we instrumented the simulator to report the results of lazy cancellation comparisons. Figures 4 and 5 contain representative plots from these studies (we examined considerably more) for two different LPs of BENCH_FPA and ADDER16. The x's in the plots indicate whether the result of the output message comparison was a *Hit* (if it is a 1) or a *Miss* (if it is a 0). The data clearly indicates the resulting comparisons vary widely between LPs and even during the lifetime of an LP.

The dynamic selection of the cancellation strategy by using a thresholding function was reviewed in the previous section. Use of a thresholding function is not the only solution to the



(a) (b)

FIGURE 4 Example where the optimal cancellation strategy varies.

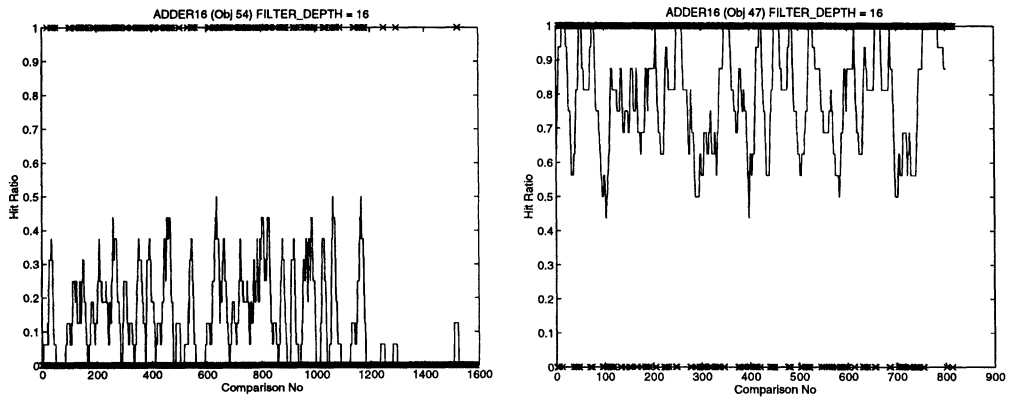


FIGURE 5 Example where LPs in the same application favor different cancellation strategies.

dynamic selection algorithm. In fact, there exists several alternative, equally plausible mechanisms for achieving the desired adaptive control. These can be seen by analyzing the plots in Figures 4 and 5. There are three other techniques which can be used to dynamically switch the cancellation strategy. Thus, the following strategies for the dynamic cancellation function were also implemented for comparative analysis:

(a) Continuously monitor the hit/miss rate as described in the earlier section. Then decide the cancellation strategy using one of the following functions:

   1. One Threshold: The upper and lower thresholds are equal. The dead zone does not exist as both thresholds are equal. LPs that have hit miss ratios greater than the threshold will choose lazy cancellation and those which have hit miss ratios lesser than the threshold will follow an aggressive cancellation strategy.
   2. Two Thresholds: A upper and lower threshold is defined and a dead zone exists in between these two thresholds. LPs whose hit miss ratio falls within the dead zone, do not change their cancellation strategy.
   3. Three Thresholds: An additional third threshold is placed below the lower threshold (in the two threshold scenario) that permanently enables aggressive cancellation. Any LP with a hit miss ratio below this third threshold has its cancellation strategy permanently set to aggressive cancellation; furthermore, the LP no longer performs output message comparison, thereby reducing overhead.

   In Figure 4(a), object number 8 exhibits prolonged unsteady behavior. In sharp contrast to this type of behavior, object number 15 (Fig. 4(b)) shows a remarkably steady behavior, with the hit ratio being zero the majority of the time. It is obvious that using any one of the above mentioned cancellation functions would be beneficial to both objects.

(b) Begin the simulation with all LPs using lazy cancellation. If n consecutive misses ever occur, permanently switch to aggressive cancellation.

(c) Monitor only the first k comparisons and depending upon the result permanently set the LP's cancellation strategy. The threshold value used to determine when to select lazy or aggressive cancellation can vary.

All the above mentioned strategies have been used in the following section to demonstrate the

usefulness of dynamic cancellation as an optimization to Time Warp.

5.3. Performance Results

This section presents the performance results of dynamic cancellation relative to aggressive and lazy cancellation on two different application domains. Figure 6 shows the plot of the execution time as a function of the number of requests for RAID for different settings namely

AC: aggressive cancellation,
LC: lazy cancellation,
DC: dynamic cancellation with FILTER_DEPTH = 16, A2L_Threshold = 0.45 and L2A_ Threshold = 0.2,
ST0.4: dynamic cancellation with a single threshold A2L_Threshold = L2A_Threshold = 0.4,
PS32: dynamic cancellation with setting the cancellation strategy permanently after FILTER_DEPTH = 32 comparisons, and
PA10: dynamic cancellation with setting the cancellation strategy permanently to aggressive if 10 successive comparisons result in misses.

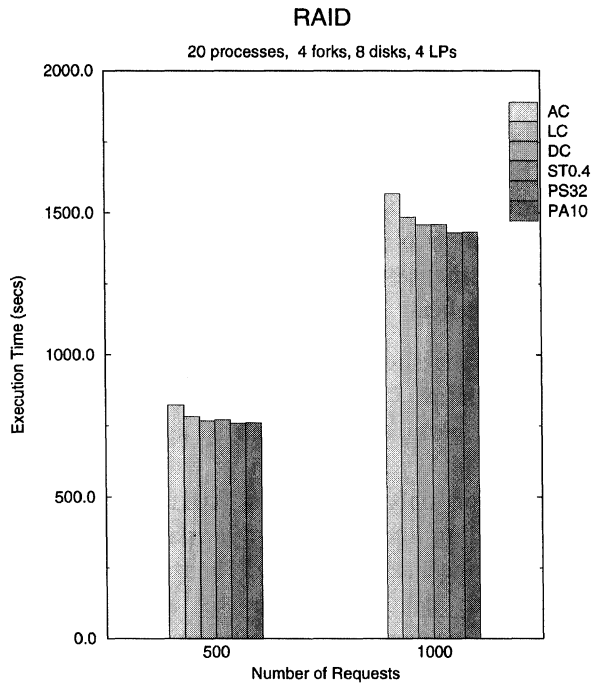


FIGURE 6 Plot of execution time *vs* number of requests for RAID.

In this application, all the disk objects favor lazy cancellation while all the fork objects favor aggressive cancellation. In the configuration chosen for this investigation, there are more disk objects than fork objects. Consequently, lazy cancellation provides better performance than aggressive cancellation (lazy cancellation is 5% faster than aggressive cancellation). Dynamic Cancellation and all its variations perform better than lazy cancellation due to the fact that all the objects strictly favor either aggressive or lazy cancellation. DC and ST0.4 perform about 1.5% faster than lazy cancellation while PS32 and PA10 provide a 2.5% speedup (because the cost of doing passive comparisons is completely avoided by the objects which favor aggressive cancellation).

Figure 7 shows the plot of the execution time as a function of the number of requests for SMMP for different settings namely

AC: aggressive cancellation,
LC: lazy cancellation,
DC: dynamic cancellation with FILTER_DEPTH = 16, A2L_Threshold = 0.45 and L2A_Threshold = 0.2,
PS: dynamic cancellation with setting the cancellation strategy permanently after FILTER_ DEPTH = 64 comparisons, and
PA: dynamic cancellation with setting the cancellation strategy permanently to aggressive if 10 successive comparisons result in misses.

In this application, all the objects having rollbacks strictly favor lazy cancellation and lazy cancellation is 15% faster than aggressive cancellation in this application. Consequently all the variations of dynamic cancellation perform on par with lazy cancellation, giving no speedup over the best cancellation strategy (*i.e.*, lazy cancellation). PS64 performs slightly better than DC and PA10 because it permanently switches into lazy cancella-

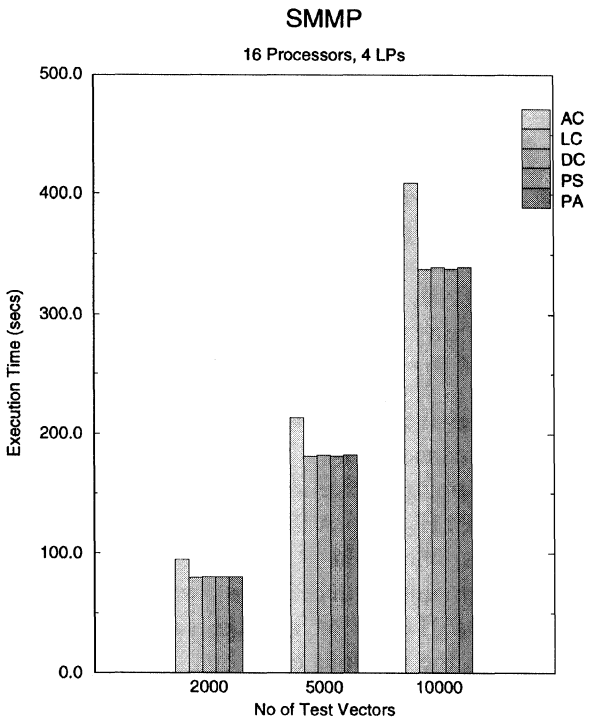


FIGURE 7 Plot of execution time *vs* number of test vectors for SMMP.

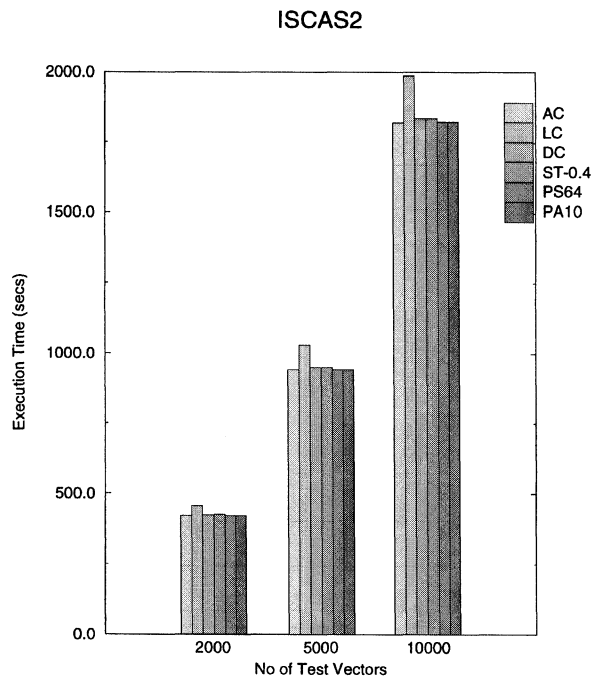


FIGURE 8 Plot of execution time *vs* number of test vectors for ISCAS2.

tion after 64 comparisons and does not have to monitor the hit (misses) throughout the simulation.

Figure 8 shows the plot of the execution time as a function of the number of test vectors for ISCAS2 for different settings namely

AC: aggressive cancellation,

LC: lazy cancellation,

DC: dynamic cancellation with FILTER_DEPTH = 16, A2L_Threshold = 0.4 and L2A_Threshold = 0.2,

ST0.4: dynamic cancellation with FILTER_DEPTH = 16 and A2L_Threshold = L2A_ Threshold = 0.4,

PS64: dynamic cancellation with setting the cancellation mode permanently after the first FILTER_DEPTH = 64 comparisons with A2L_Threshold = 0.4 and L2A_Threshold = 0.2 and

PA10: dynamic cancellation with setting the cancellation strategy permanently to aggressive if 10 successive comparisons result in misses, with FILTER_DEPTH = 16, A2L_Threshold = 0.4 and L2A_Threshold = 0.2.

This application performs better under aggressive cancellation (9% faster than lazy cancellation) because almost all objects with rollback favor aggressive cancellation throughout the simulation run. DC and ST0.4 perform slower than aggressive cancellation because they have to do message comparisons even though they are in the aggressive mode most of the time. PS64 and PA10 perform better than DC and ST0.4 because they do not have to keep track of the hit/misses after a certain number of comparisons.

Figure 9 shows the plot of the execution time as a function of the time (simulation time) for which the clock was allowed to run, for BENCH_FPA for different settings namely

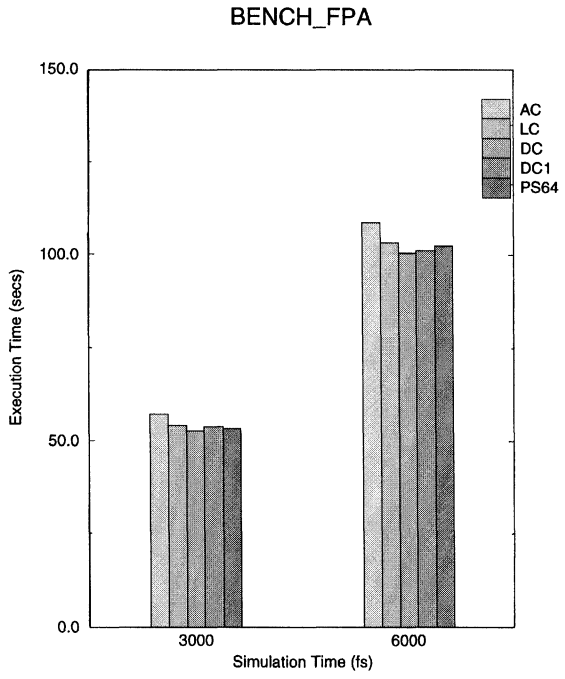


FIGURE 9 Plot of execution time *vs* simulation time for BENCH_FPA.

AC: aggressive cancellation,
LC: lazy cancellation,
DC: dynamic cancellation with FILTER_DEPTH = 16, A2L_Threshold = 0.4 and L2A_Threshold 0.2,
DC1: dynamic cancellation with FILTER_DEPTH = 16, A2L_Threshold = 0.6 and L2A_Threshold = 0.4, and
PS64: dynamic cancellation with setting the cancellation mode permanently after the first FILTER_DEPTH = 64 comparisons with A2L_Threshold = 0.4 and L2A_Threshold = 0.2.

In this application lazy cancellation performs better than aggressive cancellation (5% faster). DC performs better than lazy cancellation (2% faster) due to the fact that the favorable cancellation strategy for the some of the objects varies over the lifetime of the simulation (Fig. 4). DCl is slower than DC due to the fact that the A2L_Threshold (0.6) is too high to obtain any benefit from the switch. PS64 is slower than DC and DC1 due to the fact that the optimal cancellation strategy for some of the objects varies over the lifetime of the simulation (Fig. 4). By fixing the cancellation strategy after 64 comparisons, the advantage of using the cancellation strategy most appropriate beyond that point in the simulation is lost.

Figure 10 shows the plot of the execution time as a function of the number of test vectors for ADDER16 for different settings namely

AC: aggressive cancellation,
LC: lazy cancellation,
DC: dynamic cancellation with FILTER_DEPTH = 16 A2L_Threshold = 0.45 and L2A_Threshold = 0.2,
DC64: dynamic cancellation with FILTER_DEPTH = 64 and the same thresholds as the previous case,
DC1: dynamic cancellation with FILTER_DEPTH = 16, A2L_Threshold = 0.6, L2A_Threshold = 0.4 and

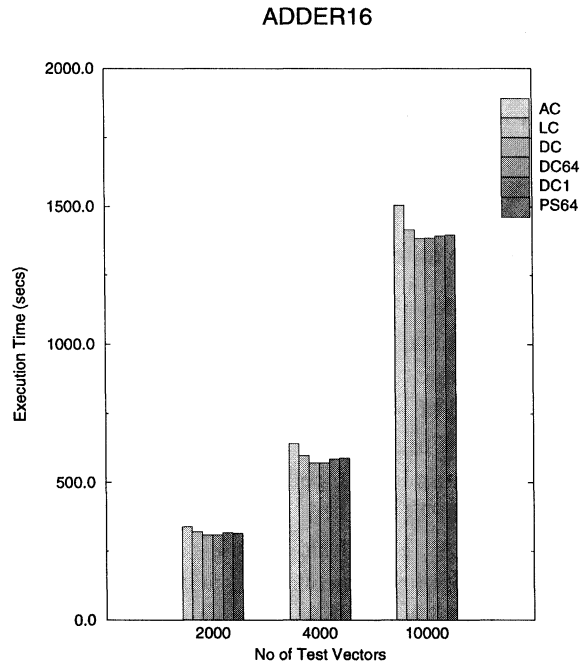


FIGURE 10 Plot of execution time *vs* number of test vectors for ADDER16.

PS64: dynamic cancellation with setting the cancellation mode permanently after the first FILTER_DEPTH = 64 comparisons with A2L_Threshold = 0.45 and L2A_Threshold = 0.2.

In this application, a majority (60%) of the simulation objects favor lazy cancellation most of the time while the remaining objects favor aggressive cancellation most of the time, thereby causing dynamic cancellation to perform better. DC and DC64 provide almost the same performance suggesting that the length of the measurement cycle (*i.e.*, FILTER_DEPTH) does not have a major influence on the performance of dynamic cancellation. Note that DC1 performs slightly worse compared to DC suggesting that the choice of the thresholds plays an important role in the performance of dynamic cancellation. PS64 performs worse than DC.

In addition to the above experiments, experiments were done using larger models. Due to time and space constraints, only the SMMP and the RAID model were tested with larger examples. Considerably larger models were chosen, and a comparison between the best static and the best dynamic strategy was done.

In the case of the RAID model, the best static strategy was the lazy cancellation strategy (LC), while the best dynamic strategy was dynamic cancellation with the cancellation strategy being set permanently to aggressive if 10 successive comparisons resulted in misses (PA10). For the SMMP model, the best static strategy was again lazy cancellation (LC), while the best dynamic strategy was dynamic cancellation with the cancellation strategy being permanently set after 64 comparisons (PS64). As seen in the smaller models, the RAID and the SMMP applications perform well with the dynamic strategies.

For the RAID model, a large collection of 1000 source processes were created to serve requests to 10 disks through four fork objects. Each source process was initialized to generate 1000 requests each. Due to the large number of requests being generated, the dynamic strategy performed 8% faster than the best static strategy.

For the SMMP model, a large 1000 processor simulation was run. This model had a total 6004 simulation objects. Each of the 1000 processors was initialized to generate 1000 requests. Again, as seen in the case of the smaller model, the dynamic strategy performs well in comparison to the best static strategy. In particular, the best dynamic strategy performs 10% faster than the best static strategy.

6. CONCLUDING REMARKS

The performance of a Time Warp simulator depends on the cancellation strategy employed by it to undo the erroneous premature work done by the LPs. Depending on the characteristics of the application being simulated, the way the application is partitioned, the nature of the input data, and the architecture on which the application is simulated (and also the way the application is modeled) either lazy or aggressive cancellation can provide better performance. In studies conducted as a part of this investigation, we have confirmed the observation that neither strategy is clearly superior to the other for all cases. Furthermore, no practical techniques for statically determining which cancellation strategy will produce the best performance results are known. Furthermore, we have observed that the favorable cancellation strategy can vary over the lifetime of the simulation and also from LP to LP.

This paper develops a Time Warp optimization, called *Dynamic Cancellation*, that lets each LP dynamically decide which cancellation strategy to employ. An implementation of dynamic cancellation that monitors the frequency at which the same output events are regenerated after rollback was implemented and evaluated. The following observations can be made from the experiments conducted to evaluate Dynamic Cancellation:

- In applications where a large percentage of the LPs favor the same cancellation strategy, dynamic cancellation performs on par with that cancellation strategy.
- Dynamic cancellation provides a performance improvement over the existing cancellation strategies for those applications where the cancellation strategy favored by a simulation object varies over the lifetime of the simulation and also from LP to LP.
- Using a single threshold to switch between cancellation strategies might not provide a performance improvement because a single hit or miss can cause the LPs to switch from one strategy to another, and the LPs may end thrashing between cancellation strategies.
- The variation of dynamic cancellation where an LP having m consecutive misses is permanently switched to aggressive cancellation provides a good performance when there are LPs which strictly favor aggressive cancellation. By setting such LPs to use aggressive cancellation permanently, the cost of doing passive comparisons is completed avoided. This variation of dynamic cancellation also lets LPs whose favorable cancellation strategy varies over the lifetime of the simulation to choose the cancellation strategy most favorable to them at that point in the simulation, thereby obtaining the best possible performance. An interesting extension to this might be to develop some kind of sweep mechanism that periodically returns all LPs to lazy cancellation.
- The idea of permanently selecting a cancellation strategy based on the first m output message comparisons works best when most of the LPs of the application attain steady state early in the simulation.
- The best general setup to obtain a consistent performance improvement is to use two thresholds separated by a dead zone, to switch between cancellation strategies.

Consistent speedups of 5%–10% over the best existing cancellation strategy have been observed from dynamic cancellation using the following settings – FILTER_DEPTH = 16, A2L_Threshold = 0.45, and L2A_Threshold = 0.2. Although the *permanently set* strategy did not give a speedup over the best default strategy, it chose the best cancellation strategy for higher values of FILTER_DEPTH (*i.e.*, 64 and above). Finally, dynamic cancellation has the same space requirements as lazy cancellation.

Acknowledgements

The authors would gratefully like to acknowledge the suggestions and contributions of David Charley, Balakrishnan Kannikeswaran, Dale E. Martin, Timothy J. MeBrayer, Lantz Moore, John Penix, and Christopher H. Young.

References

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

[2] Ball, D. and Hoyt, S. "The adaptive time-warp concurrency control algorithm", In: *Distributed Simulation*, pp. 174–177, Society for Computer Simulation, January 1990.

[3] Lin, Y.-B., Preiss, B. R., Loucks, W. M. and Lazowska, E. D., "Selecting the checkpoint interval in time warp simulation", In: *Proc of the 7th Workshop on Parallel and Distributed Simulation (PADS)*, pp. 3–10, Society for Computer Simulation, July 1993.

[4] Matsumoto, Y. and Taki, K., "Adaptive time-ceiling for efficient parallel discrete event simulation", In: *Object-Oriented Simulation Conference (OOS '93)*, (Beaumariage, T. and Roberts, C., Eds.), pp. 101–106, Society for Computer Simulation, January 1993.

[5] Palaniswamy, A. and Wilsey, P. A. (1993). "Adaptive bounded time windows in an optimistically synchronized simulator", In: *Third Great Lakes Symposium on VLSI*, pp. 114–118.

[6] Palaniswamy, A. and Wilsey, P. A. (1993). "Adaptive checkpoint intervals in an optimistically synchronized parallel digital system simulator", In: *VLSI 93*, pp. 353–362.

[7] Palaniswamy, A. and Wilsey, P. A., "Scheduling time warp processes using adaptive control techniques", In: *Proceedings of the 1994 Winter Simulation Conference* (Tew, J. D., Manivannan, S., Sadowski, D. A. and Seila, A. F., Eds.), pp. 731–738, December 1994.

[8] Reiher, P. L., Wieland, F. and Jefferson, D. R., "Limitation of optimism in the time warp operating system", In: *Winter Simulation Conference*, pp. 765–770, Society for Computer Simulation, December 1989.

[9] Astrom, K. J. and Wittenmark, B. (1989). *Adaptive Control.* Reading, MA: Addison Wesley.

[10] Palaniswamy, A. and Wilsey, P. A. (1996). "Parameterized time warp (PTW): An integrated adaptive solution to

optimistic PDES", *Journal of parallel and Distributed Computing*, **37**, 134–145.
[11] Jefferson, D. (1985). "Virtual time", *ACM Transactions on Programming Languages and Systems*, **7**, 405–425.
[12] Gafni, A. "Rollback mechanisms for optimistic distributed simulation systems", In: *Distributed Simulation*, pp. 61–67, Society for Computer Simulation, January 1988.
[13] West, D. (1988). "Optimizing time warp: Lazy rollback and lazy re-evaluation", *Master's Thesis*, University of Calgary, Calgary, Alberta.
[14] Reiher, P. L., Fujimoto, R. M., Bellenot, S. and Jefferson, D., "Cancellation strategies in optimistic execution systems", In: *Proceedings of the SCS Multiconference on Distributed Simulation*, **22**, 112–121, Society for Computer Simulation, January 1990.
[15] Das, S., Fujimoto, R., Panesar, K., Allison, D. and Hybinette, M., "GTW: a time warp system for shared memory multiprocessors", In: *Proceedings of the 1994 Winter Simulation Conference* (Tew, J. D., Manivannan, S., Sadowski, D. A. and Seila, A. F., Eds.), pp. 1332–1339, December 1994.
[16] Jefferson, D., Beckman, B., Wieland, F., Blume, L., Di Loreto, M., Hontalas, P., Laroche, P., Sturdevant, K., Tupman, J., Warren, V., Wedel, J., Younger, H. and Bellenot, S. (1987). "Distributed simulation and the time warp operating system", In: *Proceedings of the 12th SIGOPS – Symposium of Operating Systems Principles*, pp. 77–93.
[17] Martin, D. E., MeBrayer, T. J. and Wilsey, P. A., "WARPED: A time warp simulation kernel for analysis and application development", In: *29th Hawaii International Conference on System Sciences (HICSS-29)*, (ElRewini, H. and Shriver, B. D., Eds.), Vol. I, 383–386, January 1996.
[18] Lin, Y. (1996). "Estimating the likelihood of success of lazy cancellation in time warp simulations", *International Journal in Computer Simulation*.
[19] Misra, J. (1986). "Distributed discrete-event simulation", *Computing Surveys*, **18**, 39–65.
[20] Fujimoto, R. M., "Time warp on a shared memory multiprocessor", *Transactions of Society for Computer Simulation*, pp. 211–239, July 1989.
[21] Fleischmann, J. and Wilsey, P. A., "Comparative analysis of periodic state saving techniques in time warp simulators", In: *Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS 95)*, pp. 50–58, June 1995.
[22] Lin, Y.-B. and Lazowska, E. D., "The optimal checkpoint interval in time warp parallel simulation", Tech. Rep. 89-09-04, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, September 1989.
[23] Rödnngren, R. and Ayani, R., "Adaptive checkpointing in time warp", In: *Proc. of the 8th Workshop an Parallel and Distributed Simulation (PADS 94)*), pp. 110–117, Society for Computer Simulation, July 1994.
[24] Palaniswamy, A., Aji, S. and Wilsey, P. A., "Performance measures for several optimizations to a distributed digital system simulator", In: *Proc. of the 26th Annual Simulation Symposium*, pp. 21–29, Society for Computer Simulation, April 1993.
[25] Martin, D. E., McBrayer, T. and Wilsey, P. A. (1995). "WARPED": A time warp simulation kernel for analysis and application development" (available on the www at http://www.ece.uc.edu/paw/warped/).
[26] Berry, O. and Jefferson, D., "Critical path analysis of distributed simulation", In: *Distributed Simulation*, pp. 57–60, Society for Computer Simulation, 1985.
[27] Rajan, R. and Wilsey, P. A., "Dynamically switching between lazy and aggressive cancellation in a time warp parallel simulator", In: *Proc. of the 28th Annual Simulation Symposium*, pp. 22–30, IEEE Computer Society Press, April 1995.
[28] Navabi, Z. (1993). *VHDL: Analysis and Modeling of Digital Systems*, New York, NY: McGraw-Hill.
[29] Perry, D. L. (1994). *VHDL*, New York, NY: McGraw-Hill, 2nd ed.
[30] Chen, P. M. (1994). "Raid: High-performance, reliable secondary storage", *ACM Computing Surveys*, **26**, 145.

Authors' Biographies

Raghunandan Rajan is a CAD Engineer working for Intel Corporation in Chandler, AZ. He received his MS. degree in Electrical and Computer Engineering from the University of Cincinnati and his B.Tech degree in Tele communication Engineering from Jawaharlal Nehru Technological University, Hyderabad, India. His current research interests include parallel discrete event simulation, HDL co-simulation, high level synthesis and design for low power. He is a member of the IEEE Computer Society.

Radharamanan Radhakrishnan is a Ph.D. student in the Department of Electrical and Computer Engineering and Computer Science at the University of Cincinnati. He received a B.S. degree in Electrical and Computer Engineering from the University of Madras in 1993. His current research interests include parallel discrete event driven simulation, parallel processing, formal methods and networks.

Philip A. Wilsey is an Assistant Professor in the Department of Electrical and Computer Engineering and Computer Science at the University of Cincinnati. He received Ph.D. and M.S. degrees in Computer Science from the University of Southwestern Louisiana and a B.S. 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 member of both the IEEE Computer Society and the ACM.