

Time Warp Parallel Simulation of VHDL Descriptions and the need for Dynamic Parameter Adjustment*

Dale E. Martin, Tim J. McBrayer, and Philip A. Wilsey
Computer Architecture Design Lab
Dept. of ECE, PO Box 210030
Cincinnati, OH 45221-0030
dmartin,tmcbraye@ece.uc.edu phil.wilsey@uc.edu

Abstract

The time warp synchronization mechanism has the potential to speedup VHDL simulations on parallel platforms. However, practical implementations of time warp have been hindered by several drawbacks such as large memory usage, excessive rollbacks (instability), and wasted lookahead computation. These problems have been addressed by a variety of optimizations and variations to the original time warp mechanism. While many of the optimizations show good isolated performance, they remain unable to satisfy the objectives of realizing good performance against a broad range of simulation problems. Fortunately, many of the problems are addressable by the addition of dynamic parameter adjustment. A Parameterized Time Warp mechanism is presented that provides a simple integrated solution to the problems associated with the original Time Warp mechanism. The mechanism is based on parameters of useful work that represents the amount of productive work done by each LP. We use these measures to update the following parameters in a time warp simulator: (i) checkpoint interval, (ii) cancellation strategies, and (iii) bounding windows. In addition, viewed as a controllable system, new, less intrusive GVT algorithms for time warp have been developed.

1 Introduction

Simulating VHDL descriptions of digital systems can require considerable system resources. A typical contemporary workstation is generally sufficient only for brief simulations of small VHDL descriptions. While the acquisition of a larger machine with a high performance processor and huge memory can alleviate

this problem, it is an expensive option that is rarely cost effective. An alternative solution is to simulate VHDL descriptions on a parallel SMP workstation or on a distributed network of workstations. Several possibilities exist for synchronizing parallel and distributed simulations, namely: central event, conservative, or optimistic [7, 20]. This paper discusses one of these synchronization protocols, in particular, the optimistic synchronization strategy called Time Warp [9].

The Time Warp mechanism is one of the most important synchronization protocols for parallel simulation [7]. However in many applications, the successful use of Time Warp requires the careful selection of Time Warp optimization parameters (*e.g.*, cancellation strategies, frequency of state saving, and so on). Furthermore, the optimal setting for the simulation parameters may not hold 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, 16, 19, 22, 23, 25, 29]. These investigations have shown that dynamic parameter adjustment can be used to successfully used to tune a Time Warp simulator for near optimal 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 [1]. 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. In general, past studies to control Time Warp have relied on simple ad hoc control mechanisms [19, 22, 23] using relatively primitive system models.

In this paper, we examine the construction of non-

*This work was partially supported by the Defense Advanced Research Projects Agency under order number 7056, monitored by the Department of Justice under contract number J-FBI-93-116.

linear and adaptive control system to dynamically adjust simulation control parameters. This paper includes a tutorial introduction to Time Warp and a review of the basic elements of control theory. The object of this paper is to describe Time Warp and the application of non-linear and adaptive control mechanisms to achieve good time and space performance given a wide range of application demands (as one typically finds across the description space of VHDL).

The remainder of the paper is organized as follows. Section 2 presents a background discussion Time Warp. Section 3 reviews some basics aspects of control theory and present Time Warp as a system of controllable processes. Section 4 contains some empirical data to demonstrate the utility of applying control techniques to Time Warp. Finally, Section 5 contains some concluding remarks.

2 Time Warp

A Time Warp Parallel Discrete Event Simulation (PDES) contains a set of Logical Processes (LP) that operate as asynchronously communicating discrete event simulators [9, 7]. Each LP maintains its own simulation clock called the Local Virtual Time (LVT) and event information between LPs is communicated as timestamped messages. The LPs operate asynchronously, processing event messages in their timestamp order. To correctly simulate a physical system, the LPs must process the arriving messages in their timestamp order, as opposed to their real-time order (termed as the *Local Causality Constraint* [7]). Thus, to satisfy this causality constraint, the Time Warp mechanism supports synchronization with rollback processing *i.e.*, the LPs do not strictly avoid causality errors, instead they allow the possibility of causality violations from occurring and recover from it as and when it is discovered. The LPs continue to receive and process event-messages from their *input queues* until there are no more unprocessed messages, or until an event-message with a time-stamp lower than the LVT of that LP arrives. Such an out-of-order message is called a *straggler* message. When a straggler arrives at an LP, the LP discovers that it has not been processing the input messages in order, and consequently, the LP stops processing messages from its input queue and performs a *rollback* — restructuring the simulation to process the messages in proper time-stamp order. A picture illustrating the chief components in a Time Warp LP is shown in Figure 1.

A *rollback* consists of restoring the state of the LP to the exact point where the straggler should have been

processed (had it arrived in order), and cancelling the side-effects of any erroneous lookahead computation. In order to facilitate rollback, each LP maintains an *Input Queue* that contains all the processed and unprocessed event-messages, a *State Queue* that contains snapshots of the LP's state after each event-message has been processed, and an *Output Queue* that contains negative copies (referred to as *anti-messages*) of the messages sent out by the LP. The purpose of an anti-message is to curtail the spread of the erroneous computation by annihilating the corresponding positive message from the input queue of the receiving process. The time at which the antimessages are generated can vary between two strategies: *aggressive cancellation* and *lazy cancellation*.

The *Aggressive Cancellation* (AC) strategy causes the immediate generation of antimessages upon receipt of a straggler message. The assumption made in aggressive cancellation is that messages arriving out-of-order always produce erroneous output messages and consequently, antimessages are immediately dispatched to avert cascading rollbacks. Studies have shown that, in general not all the out-of-order messages generate erroneous output [10, 11, 28] and thus, lazy cancellation is suggested to delay the generation of antimessages.

The *Lazy Cancellation* (LC) strategy delays the sending of antimessages until recomputation discovers that the earlier message was actually incorrect. This requires that on rollback output messages be compared previously sent output messages to determine if the earlier lookahead computation was incorrect. If the earlier message was incorrect (called a *lazy miss*) then its antimessage is sent. While this has the potential effect of allowing erroneous messages to enter the system for longer time periods, it also has the potential to allow the simulation to perform better than the critical path [10].

The chief problem is that neither cancellation strategy is clearly superior to the other for all applications [28]. Furthermore, in several local studies, we have seen that different LPs within the same application may operate best with different cancellation strategies and furthermore, that the ideal cancellation strategy may vary over the lifetime of the simulation.

Another important component of a Time Warp simulation is the checkpoint frequency (*i.e.*, the number of events processed between successive state saves). Thus to process a straggler event, the Time Warp simulator returns to a state preceding the timestamp of the straggler, *coasts forward* to the straggler (processing events, but not saving state or sending output

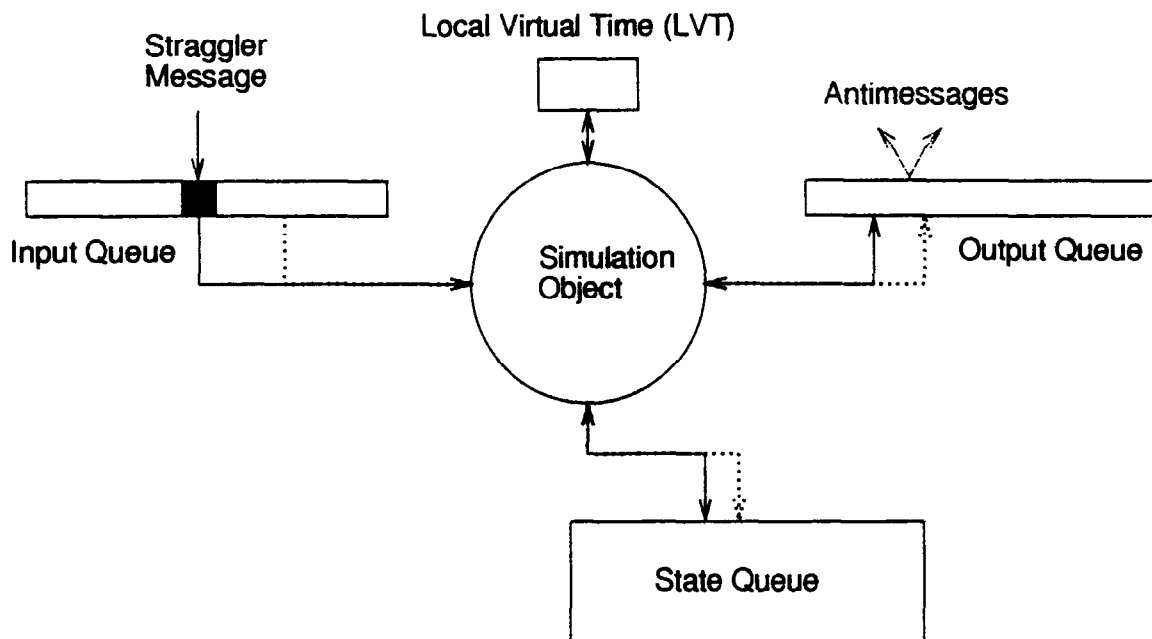


Figure 1: Organization of a Logical Process in a Time Warp simulation

messages). Upon reaching the straggler, the simulation continues as before — processing events, generating output events, and saving state; applying the appropriate event cancellation strategy.

In principle, a Time Warp simulator can use any value for the checkpoint frequency. However for optimal performance, there is a breakpoint at which the coast forward costs exceed the state savings cost. Of course, this cost is a function of rollback frequency [6] and, thus, can only be optimally determined at runtime.

Finally, a Time Warp simulation can consume considerable memory to save old event and state information. This memory must be pruned (and reused) when it is no longer needed (*i.e.*, when global progress of the simulation advances such that no rollback to the timestamps of the saved information can occur). Such memory is called a *fossil* and it is reclaimed using a global time value called *Global Virtual Time* or GVT. GVT is defined as the minimum of (a) the smallest local virtual time of all the Time Warp processes in the system and (b) the smallest timestamped message in transit. Therefore the storage used by state, input, and output queues below this time can be reclaimed. Reclaiming of these storage locations is commonly referred to as *fossil collection* [9]. When all messages have been processed by a processor, LVT is set to in-

finity (∞) and a GVT value of ∞ is a sufficient condition to terminate the simulation. A higher GVT computation frequency produces faster response time and better space utilization but it also uses processor time and network bandwidth. Several GVT algorithms have been proposed to reduce the overhead of GVT computation (especially the number of messages required) [4, 5, 8, 13].

Time Warp and its Limitations

The primary role of the Time Warp mechanism is to assure that the local mechanisms maintain the global correctness of the distributed simulation. Global causality can be maintained by preserving local causality in each LP, *i.e.*, each LP should process events in timestamp order. Thus, LVT of each LP is advanced or retracted depending on the timestamp of the next event to be processed. In Time Warp, a causality error is detected by an LP when it receives an event with a timestamp smaller than that of the last processed event (a straggler event).

When an LP receives a straggler event, the LP rolls back to an earlier virtual time, cancels all previous effects (assuming aggressive cancellation), and executes forward maintaining the causality of the simulation. Specifically, on receipt of a straggler, the current state of the LP is replaced by a state saved before the straggler timestamp and all intermediate states saved dur-

ing the lookahead computation are discarded. To undo previous global effects due to the local LP, messages sent by the LP due to lookahead must be cancelled by antimessages. Upon receipt of an antimessage, an LP must perform one of two actions. If the timestamp is larger than LVT, then this implies that the corresponding positive message has not been processed, therefore the positive message can just be cancelled. If the timestamp of the antimessage is less than or equal to LVT, then the LP cancels the positive message and also rolls back to a time earlier than the timestamp of the antimessage, and sends out its own antimessages to cancel the effects of its lookahead computation on other processes in the system. Repeating this procedure in all the communicating processes allows all the erroneous computation caused by causality errors to be cancelled.

More details about the Time-Warp mechanism can be found in [7, 9]. Some of the limitations of Time Warp are briefly discussed below.

Memory Usage

Memory is used by Time Warp for input queues, output queues, and state queues. States are saved after execution of each event in the state queue. Large state queues become a memory bottleneck [12], and can prevent the simulation from completing due to memory exhaustion. Several memory management algorithms have been suggested to manage memory in Time Warp, namely: fossil collection [9], message sendback [8], cancelback [8], artificial rollback [12], periodic state saving [14], incremental state saving [3], and rollback relaxation [31]. Despite these mechanisms, state saving still remains a bottleneck in the time and space domain. Periodic state saving [14, 16, 24] has been proposed as an efficient mechanism to reduce rollback and memory overheads. However, it has been shown that a mechanism to dynamically determine an appropriate checkpoint interval can lead to near-optimal execution times [6, 23, 30].

Stability

A *perfectly stable* Time Warp simulation is defined as a simulation which executes in near-optimal completion time. Excessive number of rollbacks causes the simulation to execute longer. Thus, an increase in the number of rollbacks would cause a decrease in stability of the system, that is longer execution time.

Instability in Time Warp can occur due to various reasons, namely:

- Cascading of rollbacks eventually causes a thrashing effect resulting in significant deterioration of the parallel simulation [17].

- Fast and slow processes in the system can result in an instable system. Consider the following scenario: A is a slow process and B is a fast process, both residing on the same processor, and A causes B to rollback. If A continues to cause B to rollback, processor bandwidth is being wasted in the lookahead computation at B. If similar situations exist in the rest of the system, not only is there an excessive amount of wasted lookahead computation, but also an inherent instability in the system [22].

- Process Scheduling is another important factor that can influence the stability of Time Warp. In particular, excessive rollbacks eventually result in the deterioration of a simulation. Consequently processes should be scheduled such that there is a fair policy to maintain the balance in LVT advancement among processes as well as CPU time provided to each process [15, 25]. Typical policies are based on smallest LVT processes first, but such simplistic policies do not account for messages arriving in the past from processes managed by other processors in a multi-processing environment.

These problems impede the use of Time Warp in a general purpose environment. Thus, there is a need to speedup and stabilize the Time Warp synchronization algorithm, and provide a practical optimistic synchronization protocol. The solutions presented in this paper are based on non-linear and adaptive control techniques that have been shown to effectively control stability and memory problems associated with the original Time Warp mechanism [22, 25].

Adaptive techniques [21] attempt to adjust the optimistic simulation towards a conservative execution strategy, based on several observations such as rollback behavior and antimessage counts. In contrast, the Parameterized Time Warp approach attempts to control and exploit optimism in ways that improve the probability of productive lookahead. The Time Warp simulation is adaptively controlled by perturbing control parameters so as to maximize “productivity” of each LP and thereby maximize overall throughput. Thus, the Parameterized Time Warp approach is based on a measure for each LP, called *useful work*, which reflects the measured “productivity” of the LP. This measure is discussed in detail below. But first, a brief introduction to control systems techniques is necessary to elaborate on the solution.

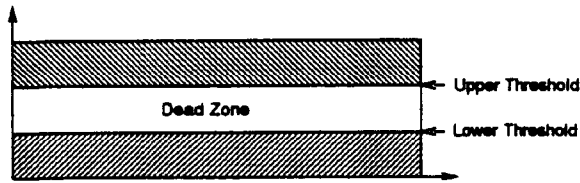


Figure 2: Thresholding to select cancellation strategies

3 Control Theory

In general, a control system samples output values and adjusts inputs in order to meet some performance criteria [1]. In a Time Warp system, the outputs can be a variety of values 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, 16, 19, 22, 23, 25, 29]. In general, these investigations must use data filtering techniques to smooth and to prevent spurious data points from causing wide variations parameter adjustment. Furthermore, the controller adjusts the input parameters based on some model of the simulation that projects how parameter changes will affect output response.

In this paper, we will explore two techniques for controlling the parameters of a Time Warp simulation. In particular, we will use a non-linear thresholding function to control cancellation strategies and an adaptive control model to set checkpoint frequency.

3.1 Cancellation Strategies

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.

In the experiments described below, we will use a thresholding function to decide cancellation strategies. The thresholding function will be computed on each LP and will monitor the lazy hit/miss ratio. When the lazy hit/miss rate falls above the upper threshold, the lazy cancellation strategy will be employed by that LP.

A hit/miss rate below the lower threshold will cause the use an aggressive cancellation strategy. As the rate moves within and between these two thresholds, the cancellation strategy remains unchanged; when it crosses the upper or lower threshold, a switch to the alternate cancellation strategy occurs. The hit/miss ratio is revised on every valid comparison (*i.e.*, a valid comparison is made only when there is an event previously placed in the output queue whose destination and timestamp matches that of the event being generated) and is computed as a ratio of the last 16 valid comparisons. The upper threshold is set at .4 and the lower threshold is set at .2.

3.2 Adaptive Control Theory

Control theory is concerned with modifying the behavior of dynamic systems so as to achieve desired goals [1]. These goals include maintaining outputs at constant levels, assuring that the overall system track specified trajectories, or more generally the overall system optimizes a set of specified criteria. The goal is achieved by finding suitable control inputs based on the observed outputs of the system. The fundamental processes involved in *controlling* a dynamic system include:

1. mathematical modeling of the system,
2. identification of the system based on experimental data, and
3. processing outputs and using them to synthesize control inputs to achieve the desired behavior.

Any adaptive system, no matter how complex, is merely a feedback system involving estimation and control. A designer who sets out to control a system with a certain degree of uncertainty and produces a controller which achieves its objective may conclude that he/she has designed an adaptive system. Adaptive control provides potential solutions to problems of the following general form. *The system to be controlled is normally exposed to a time-varying environment, in the form of a system with changing parameters, input signals, disturbances with time-varying statistical characteristics, or changing performance objectives.* However, adaptive control is not easy for the following reasons:

- Most systems have transfer functions which are difficult or impossible to determine.
- Parameter estimation is difficult.
- The adaptive element should be as simple as possible, so as not to affect the performance of the system.

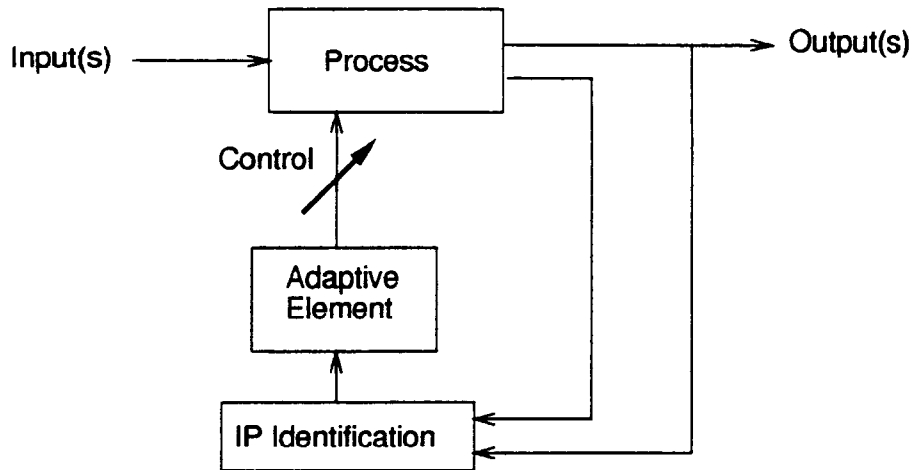


Figure 3: Adaptive control for a Time Warp LP

To overcome these problems, control theorists have developed a system control technique called *Model Reference Adaptive System (MRAS)*. A MRAS builds a model of the expected behavior and samples the outputs to compare against the expected response. The model adjusts a set of control inputs to lead the system into a desired state.

In the work described in this paper we build an LP reference model that monitors outputs and adjusts inputs based on the desired output; in particular, we will attempt to find the knee of the curve that balances checkpoint costs against coast forward costs to find the optimal checkpoint frequency [6]. The experiments described in this paper places a low-pass filter on each sampled output data line. More precisely, we use a first order IIR filter with coefficients $\alpha_0 = .8$ and $\alpha_1 = .2$ on each data line. Additional details regarding the treatment of a Time Warp simulation as a controllable system is described below.

3.3 Time Warp as a System of Controllable Processes

We consider a Time Warp simulation as a system of processes as shown in the block diagram of Figure 4. The complete Time Warp system is viewed as a system of processes organized as blocks interconnected as determined by the communication structure of the simulation. The Time Warp simulation is near-optimal if the output at each block is near-optimal. The output at each block can be made near-optimal by adaptively controlling the process using the non-linear thresholding model for cancellation strategies and the MRAS model for selecting cancellation strategies.

In general, the input, output, and control parameters of Time Warp can be described as:

- *Inputs:* (i) Events in the past (present), (ii) Events in the future, (iii) Antimessages received, and (iv) GVT input.
- *Outputs:* (i) Positive messages sent, (ii) Negative messages sent, (iv) Rollbacks, and (iv) GVT information sent.
- *Control Parameters:* (i) Bounded Time Window size, (ii) State Saving Period (assuming periodic state saving is used), (iii) Cancellation strategy, and (iv) Priority for scheduling purposes.

In this paper, we describe our work with checkpoint frequencies, cancellation strategies, and scheduling.

4 Empirical Results

This section contains empirical performance results using a collection of diverse queuing models of systems. These examples are all executed on an unloaded 4 processor SUN SparcCenter 1000. Each example was executed multiple times and the performance figures averaged. The examples are selected to represent various modeling problems and are instantiated on the queuing model library released with the (public domain) WARPED Time Warp simulation kernel [18].

Three examples of system models are used in this study. The examples are:

RAID: This is a simulation of a nine disk RAID level 5 disk array of IBM 0661 3.5" 320MB SCSI disk

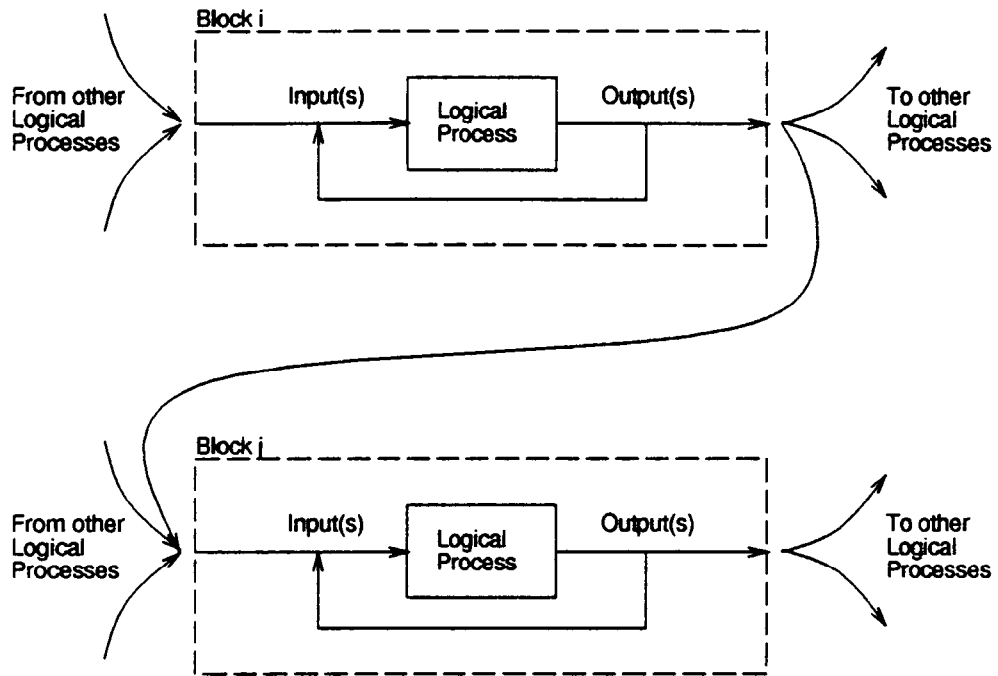


Figure 4: Time Warp as a control system

drives with a flat-Left Symmetric parity placement policy. Sixteen processes send requests for stripes of random lengths and location to forks which split the requests into individual requests for each disk according to the placement policy. The nine servers process the requests in a FCFS fashion and route the appropriate requests back to the source process.

SMMP: The SMMP model is designed to simulate several processors, each with their own cache, and sharing a 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 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 numbers used for the experiments of this paper are: simulate 16 processors on 4 LPs. The cache speed was set as 10ns and the main memory for 100ns. The cache hit ratio was set as 90%. Note that the generation program parti-

tions the model to take advantage of the fast intra-lp communication, and that the cache hit ratio will mainly affect the number of requests which must traverse MPI to be fulfilled, *i.e.*, for this data, only 10% of the requests had interaction with MPI.

Police Dispatch: The Police Dispatch Queuing model models a simple traffic police telecommunications network. Each district contains many counties. Each county has one or more police stations. Each police station has a fixed amount of resources. Resources such as traffic police cars, motorcycles, etc. are all centrally controlled from the station headquarters. The queuing model attempts to model a set of simple police stations coexisting within a given city. Each police station receives a random number of calls and each of these calls are routed by the telecommunication network to the nearest police car or motorcycle. Each car may take a random amount of time to process a call and sends back a message when the job is done.

To model this system, the sources have been set up to produce the random number of calls, with varying priorities. A fork object is then set up to

	Num Proc	Total Time (sec)	Events proc/sec	Events com/sec
Static	1	183.267	1425.367	1425.367
	2	120.424	1809.821	1451.471
	4	196.587	1198.304	889.133
Periodic	2	127.995	1664.580	1365.615
Checkpointing	4	142.610	1589.383	1232.669
Dynamic	2	124.053	1717.499	1412.444
Checkpointing	4	141.183	1669.641	1245.114
Lazy	2	121.223	1785.816	1445.039
Cancellation	4	139.782	1645.912	1221.011
Dynamic	2	125.947	1681.477	1392.188
Cancellation	4	140.821	1673.819	1216.817

Table 1: Performance results for RAID-5 Disk (41 time warp objects; 1,000 requests/request process)

route these incoming calls to one of the three call operators, depending upon the priority of the call. The call operators are modeled with a server. The server receives the call and decides which car to contact to process the call. Once it has decided which car to call, it sends a message to that car. The time taken to process the call is random. After a random amount of time, the car will send a message back to the station to inform them about the completed job. The sink object is present to receive the job completion messages and it is here that we collect statistics about the number of call processed by the car, the time taken to process the call and how many priority calls it received in a 24 hour shift.

These examples were chosen because that represent wide ranges of processing loads. For example, a single processor can only process approximately 1500 events per second on the RAID model and 2900 events per second for the SMMP model. In addition, these examples span a range of modeling problems that vary in the number of parallel components from 41 to 100. The examples are all executed on 1, 2, and 4 processor configurations and are executed with various optimizations turned on. The performance results from these runs are shown in Tables 1, 2, and 3. The single processor configurations are run as a single LP with

	Num Proc	Total Time (sec)	Events proc/sec	Events com/sec
Static	1	452.997	2192.350	2192.350
	2	324.297	3598.627	3063.323
	4	162.122	7115.450	5756.128
Periodic	2	338.017	3462.481	2935.909
Checkpointing	4	158.421	7254.103	5875.412
Dynamic	2	294.140	7628.972	3472.801
Checkpointing	4	137.282	11777.249	7123.627
Lazy	2	316.631	5034.827	3134.165
Cancellation	4	148.860	8538.909	6188.002
Dynamic	2	322.665	4777.403	3075.933
Cancellation	4	135.086	8052.117	5195.358

Table 2: Performance results for SMMP (100 time warp objects; 10,000 test vectors/processor)

lowest timestamp first scheduling and thus, represent a sequential simulation; this configuration runs on the basic WARPED kernel and has been crafted to eliminate most (unfortunately, not all) of the overhead of time warp (one state save at the beginning cannot be eliminated). For the 2 and 4 processor configurations, an equivalent number of heavyweight processes are started (MPI prevents threads and the shared memory implementation of MPI does not operate correctly on a SparcCenter 1000).

The results are tabulated and shown using three performance factors, namely: the total execution time, the total events processed per second, and the total events committed per second. While the first two are interesting, the commitment rate is the most important measure as it provides an measure of the amount of useful work actually performed. The tables show performance results with various optimizations turned on. More precisely, the rows denote the following configurations for the WARPED kernel:

Static: No optimizations; checkpointing taken every event and aggressive cancellation employed on rollback.

Periodic Checkpointing: Distinct from **static** in that checkpointing occurs only after processing three events (a checkpoint interval of 3). The

	Num	Total	Events	Events
	Proc	Time (sec)	proc/sec	com/sec
Static	1	151.057	2904.844	2904.844
	2	152.358	2876.575	2876.531
	4	83.264	5270.208	5270.196
Periodic	2	147.929	2962.592	2946.202
Checkpointing	4	87.907	4998.537	4988.975
Dynamic	2	155.721	2820.366	2819.962
	4	79.117	5554.725	5554.422
Lazy	2	163.586	2677.855	2669.610
Cancellation	4	87.713	4997.611	4997.600
Dynamic	2	155.923	2809.920	2804.756
	4	63.571	6892.580	6892.564

Table 3: Performance results for Police Dispatch (96 time warp objects; 24,000 tokens processed)

number three is the default configuration for the WARPED kernel [18].

Dynamic Checkpointing: Each time warp object dynamically determines its own checkpoint interval. This calculation follows the dynamic checkpointing strategy proposed by Preiss *et al* [26].

Lazy Cancellation: Distinct from static in that lazy cancellation is used by all time warp objects.

Dynamic Cancellation: Each time warp object dynamically selects its cancellation strategy. The cancellation strategy selected may change throughout the lifetime of each time warp object and follows the strategy outlined by Rajan [27].

The interesting aspects of the results is that dynamic parameter adjustment is useful. In all cases, the performance is increased. Furthermore, we have performed separate results to further confirm the utility of dynamic selection over static selection — even when the static result is optimally set [6, 22, 23, 25, 27]. This result is due to the fact that the optimal value changes over the lifetime of the simulation — dynamic adjustment responds to these changes, a static selection cannot.

5 Conclusions

The design of a Time Warp synchronized parallel simulator has been explored with specific consideration given to the dynamic selection of simulation parameters. In particular, the WARPED Time Warp simulation kernel was extended to include dynamic selection of checkpoint intervals and event cancellation strategies. Three examples were used to evaluate the performance of these extensions. In each case, the dynamic selection was shown to provide more effective performance. The examples are similar to modeling problems that would be programmed in VHDL and while no results are available for this paper, preliminary performance studies have shown similar performance gains for a VHDL simulation kernel that we are currently developing. This new kernel will operate on the WARPED simulation kernel.

Acknowledgments

The authors would like to thank Balakrishnan Kanikeswaran, Radharamanan Radhakrishnan, Raghunandan Rajan, and Christopher Young for their efforts to implement the parallel simulator and the applications, and for their help in data collection and verification.

References

- [1] ASTROM, K. J., AND WITTENMARK, B. *Adaptive Control*. Addison Wesley, Reading, MA, 1989.
- [2] BALL, D., AND HOYT, S. The adaptive time-warp concurrency control algorithm. In *Distributed Simulation* (January 1990), Society for Computer Simulation, pp. 174–177.
- [3] BAUER, H., AND SPORRER, C. Reducing rollback overhead in time-warp based distributed simulation with optimized incremental state saving. In *Proc. of the 26th Annual Simulation Symposium* (April 1993), Society for Computer Simulation, pp. 12–20.
- [4] BAUER, H., SPORRER, C., AND KRODEL, T. H. On distributed logic simulation using time warp. In *VLSI 91* (Edinburgh, Scotland, August 1991), A. Hallas and P. B. Denyer, Eds., IFIP TC 10/WG 10.5, pp. 127–136.
- [5] BELLENOT, S. Global virtual time algorithms. In *Distributed Simulation* (January 1990), Society for Computer Simulation, pp. 122–127.
- [6] 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)* (June 1995), pp. 50–58.
- [7] FUJIMOTO, R. Parallel discrete event simulation. *Communications of the ACM* 33, 10 (October 1990), 30–53.

- [8] GAFNI, A. Rollback mechanisms for optimistic distributed simulation systems. In *Distributed Simulation* (January 1988), Society for Computer Simulation, pp. 61-67.
- [9] JEFFERSON, D. Virtual time. *ACM Transactions on Programming Languages and Systems* 7, 3 (July 1985), 405-425.
- [10] JEFFERSON, D., AND REIHER, P. L. Supercritical speedup. In *Proceedings of the 24th Annual Simulation Symposium* (April 1991), A. H. Ruten, Ed., IEEE Computer Society Press, pp. 159-168.
- [11] LIN, Y., AND LAZOWSKA, E. D. A study of time warp rollback mechanism. *ACM Transactions on Modeling and Computer Simulation* 1, 1 (January 1991), 51-72.
- [12] LIN, Y.-B. Memory management algorithms for optimistic parallel simulation. In *6th Workshop on Parallel and Distributed Simulation* (January 1992), Society for Computer Simulation, pp. 43-52.
- [13] LIN, Y.-B., AND LAZOWSKA, E. Determining the global virtual time in a distributed simulation. Tech. Rep. 90-01-02, Department of Computer Science and Engineering, University of Washington, Seattle, Washington, 1989.
- [14] LIN, Y.-B., AND LAZOWSKA, E. Determining the global virtual time in a distributed simulation. In *1990 International Conference on Parallel Processing* (1990), pp. III-201-III-209.
- [15] LIN, Y.-B., AND LAZOWSKA, E. D. Processor scheduling for time warp parallel simulation. In *Advances in Parallel and Distributed Simulation* (January 1991), Society for Computer Simulation, pp. 11-14.
- [16] 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)* (July 1993), Society for Computer Simulation, pp. 3-10.
- [17] LUBACHEVSKY, B. D., ET AL. Rollback sometimes works...if filtered. In *Winter Simulation Conference* (December 1989), Society for Computer Simulation, pp. 630-639.
- [18] MARTIN, D. E., MCBRAYER, T., AND WILSEY, P. A. WARPED: A time warp simulation kernel for analysis and application development, 1995. (in preparation, additional documentation available on the www at <http://www.ece.uc.edu/~paw/warped/>).
- [19] MATSUMOTO, Y., AND TAKI, K. Adaptive time-ceiling for efficient parallel discrete event simulation. In *Object-Oriented Simulation Conference (OOS '93)* (January 1993), T. Beaumariage and C. Roberts, Eds., Society for Computer Simulation, pp. 101-106.
- [20] MISRA, J. Distributed discrete-event simulation. *Computing Surveys* 18, 1 (March 1986), 39-65.
- [21] PALANISWAMY, A. *Dynamic Parameter Adjustment to Speed Time Warp Simulation*. PhD thesis, Dept of ECE, University of Cincinnati, Cincinnati, OH, April 1994.
- [22] PALANISWAMY, A., AND WILSEY, P. A. Adaptive bounded time windows in an optimistically synchronized simulator. In *Third Great Lakes Symposium on VLSI* (1993), pp. 114-118.
- [23] PALANISWAMY, A., AND WILSEY, P. A. Adaptive checkpoint intervals in an optimistically synchronized parallel digital system simulator. In *VLSI 93* (September 1993), pp. 353-362.
- [24] PALANISWAMY, A., AND WILSEY, P. A. An analytical comparison of periodic checkpointing and incremental state saving. In *Proc. of the 7th Workshop on Parallel and Distributed Simulation (PADS)* (July 1993), Society for Computer Simulation, pp. 127-134.
- [25] PALANISWAMY, A., AND WILSEY, P. A. Scheduling time warp processes using adaptive control techniques. In *Proceedings of the 1994 Winter Simulation Conference* (December 1994), J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Eds., pp. 731-738.
- [26] PREISS, B. R., MACINTRYE, I. D., AND LOUCKS, W. M. On the trade-off between time and space in optimistic parallel discrete-event simulation. In *6th Workshop on Parallel and Distributed Simulation* (January 1992), Society for Computer Simulation, pp. 33-42.
- [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* (April 1995), IEEE Computer Society Press.
- [28] 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* (January 1990), vol. 22, Society for Computer Simulation, pp. 112-121.
- [29] REIHER, P. L., WIELAND, F., AND JEFFERSON, D. R. Limitation of optimism in the time warp operating system. In *Winter Simulation Conference* (December 1989), Society for Computer Simulation, pp. 765-770.
- [30] RÖNNGREN, R., AND AYANI, R. Adaptive checkpointing in time warp. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)* (July 1994), Society for Computer Simulation, pp. 110-117.
- [31] WILSEY, P. A., AND PALANISWAMY, A. Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation. In *International Conference on Simulation and Hardware Description Languages* (January 1994), Society for Computer Simulation, pp. 143-148.