Clustered Time Warp and Logic Simulation

*Hervé Avril *and Carl Tropper*
School of Computer Science
McGill University, Montréal, Canada H3A 2A7
Email : herve@cs.mcgill.ca, carl@magic.cs.mcgill.ca

Abstract

We present, in this paper, a hybrid algorithm which makes use of Time Warp between *clusters* of LPs and a sequential algorithm within the cluster. Time Warp is, of course, traditionally implemented between individual LPs. The algorithm was implemented in a digital logic simulator, and its performance compared to that of Time Warp.

Resting upon this platform we develop a family of three checkpointing algorithms, each of which occupies a different point in the spectrum of possible trade-offs between memory usage and execution time. The algorithms were implemented on several digital logic circuits and their speed, number of states saved and maximal memory consumption were compared to those of Time Warp. One of the algorithms saved between 35 and 50% of the maximal memory consumed by Time Warp (depending upon the number of processors used), while the other two decreased the maximal usage up to 30%. The latter two algorithms exhibited a speed comparable to Time Warp, while the first algorithm was 30-60% slower.

These algorithms are also simpler to implement than optimal checkpointing algorithms.

1 Introduction

Two diametrically opposed classes of algorithms have been developed for the synchronization of parallel simulations-the conservative approach inspired by Chandy, Misra [5], and Bryant [4], and the optimistic approach pioneered by Jefferson [12]. The two approaches differ fundamentally in how they maintain causality in the simulations. The conservative method relies upon a process to block until such time that it knows that events may be executed in strict timestamp order. In the optimistic approach processes are under no such restriction. A process is free to execute events as they arrive at the process. The consequence of making use of blocking in the conservative approach is the possible formation of deadlocks [11, 16]. In the optimistic approach messages may arrive which are not in strict timestamp order (i.e. stragglers). The process is then obliged to roll back to a state before the straggler arrived, and to cancel the effect of messages which should not have been sent. This cancellation is accomplished by the use of so-called antimessages, which "annihilate" the incorrectly sent messages upon meeting them in input queues. The interested reader may wish to consult [7] or [16] for surveys of work in parallel simulation.

Experience with Time Warp over the last several years has revealed two fundamental problems with the Time Warp paradigm–that of memory management resulting from the necessity to save states [7] and of unstable behavior [15]. A closely related problem is that of Time Warp running out of memory. In this case fossil collection can simply not liberate enough space for the simulation to continue. Approaches to this problem include the cancelback protocol and the use of artificial rollback.

Time Warp's unstable behavior is a consequence of low computational granularity; antimessages are not capable of annihilating previously sent messages quickly enough. This can lead to a series of cascading rollbacks, a devastating effect.

An application area such as logic simulation poses a severe challenge to Time Warp for precisely these reasons. In a logic simulation the number of LPs is extremely large–a small circuit will have several tens of thousands of gates. This cannot help but have an influence on the problem of memory management. The computational granularity of a logic simulation is low, bringing with it the possibility of long rollbacks. The presence of feedback loops in circuits brings with it an increased possibility for rollbacks. Nevertheless, a good deal of effort has gone into parallel logic simulation because reducing the time of uniprocessor simulators can have a significant impact on the design of VLSI systems. The interested reader might wish to consult [1] for a survey of attempts in this area. [3] describes an optimistic protocol for logic simulation, in which an upper bound on the simulator's time advance was added to Time Warp. Promising results were obtained.

With these difficulties in mind, we feel that a hybrid algorithm would be appropriate for logic simulation. In this approach we make use of Time Warp between clusters of LPs belonging to different processors and use a sequential algorithm within the clusters. In our approach, a process known as the cluster environment (CE) is associated with each cluster. A motivation for its development was the notion that a circuit could be partitioned prior to the simulation so that different functional units would reside on different processors.

*also with the Hutchison Avenue Software Corporation, Montréal, Canada

0-8186-7120-3/95 $04.00

We describe several checkpointing algorithms which are a natural consequence of this approach, in that they apply to all of the LPs in a cluster instead of individual LPs. Each of these algorithms occupies a different spot in the space-time continuum of tradeoffs inherent in checkpointing [17].

We feel that the clustering of LPs has the potential to reduce the number of rollbacks inherent in Time Warp. This is the subject of on-going research and is not discussed in this paper.

In what follows, section 2 contains a description of results related to those in our paper, section 3 contains a description of our clustering and checkpointing algorithms, section 4 describes our experiments and results, and the conclusion follows.

2 Related results

A number of attempts have been made to combine the optimistic and conservative approaches. [21] allows a process to proceed optimistically but avoids sending potentially erroneous messages to other LPs. [15] employs a window protocol to prevent LPs from getting too far apart in simulated time.

In [19] the authors present an algorithm in which LPs are grouped into clusters, Time Warp is used within the clusters and a conservative algorithm is used between clusters. By way of comparison, our algorithms make use of Time Warp between clusters and a sequential algorithm within the cluster. We feel that Clustered Time Warp (CTW) is more appropriate for massive fine grained simulations such as digital logic simulation than Local Time Warp for two reasons: first, CTW takes advantage of the fact that LPs in a cluster share the same address space, thereby making their synchronization and scheduling straightforward;second, by using a conservative algorithm between clusters, Local Time Warp might reduce the parallelism of the simulated model. In any event, as no performance results were presented for Local Time Warp, it is difficult to compare it to CTW.

To date, work on determining optimal checkpointing intervals has centered around determining intervals for individual LPs. In our approach LPs are grouped into clusters and the checkpoints are based on the inter-arrival times of messages to the clusters. No attempt is made to seek an optimal interval for individual LPs. However, as we shall see, substantial memory savings are obtained.

In [17], the authors make the important point that in selecting a checkpoint interval for Time Warp, it is important to distinguish between memory-optimal checkpoints and time-optimal checkpoints. Frequent checkpointing results in faster simulations and a larger consumption of memory.

Several algorithms for checkpointing have appeared in the literature. In [14], Lin develops upper and lower bounds for checkpoint intervals given by the following two expressions:

$$\chi^- = \lceil (2\alpha_1 + 1)\delta_s/\delta_e{}^{1/2} \rceil$$

$$\chi^+ = \lfloor (\alpha_1 - 1)\delta_s/\delta_e{}^{1/2} \rfloor$$

where α_1 is the average number of events executed in forward execution between rollbacks for $\chi = 1$, δ_s is the average time to save one state of the LP and δ_e is the average execution time of an event in normal forward execution. Experimental results indicate that the optimal length is actually closer to χ^+. In [14], the authors present an algorithm which incorporates the effects of rollback behavior on the length of the checkpoint interval. An iterative procedure in which N events are executed in each iteration determines the eventual value of χ. The value of N is selected empirically and has to be large enough for the simulation to have reached steady state. The number of iterations depends on the model simulated. In addition, the authors do not address the question of whether the iterations actually converge.

In [20] the authors develop an adaptive checkpointing algorithm based on the use of exponential smoothing. The authors derive an expression for the value of χ_{min}, the checkpoint interval which minimizes the execution time of the simulation. The expression is:

$$\chi_{min} = \lceil 2(R_{obs}/k_{obs})(\delta_s/\delta_c)^{1/2} \rceil$$

where k_{obs} is the number of rollbacks and R_{obs} the number of events executed. They make use of this expression in calculating the estimate for χ after the nth observation period.

This expression is given by

$$\begin{aligned}\chi_n &= \chi_{initial} \text{ if } n = 0 \\ &= \lceil (1-\rho)\chi_{n-1} + \rho\chi max \rceil \text{ if } k_{obs} = 0 \\ &= max(1, \lceil (1-\rho)\chi_{n-1} + \\ &\quad \rho min(\chi min(k_{obs}, R_{obs}, \delta_s, \delta_c), \chi_{max}) \rceil) \text{ otherwise,}\end{aligned}$$

where $\chi_{initial}$ is the initial value for χ, χ_{max} is an upper bound on the estimate for χ_n, and ρ is a parameter which determines the relative weight of past values of χ_n and the computed minimum value of χ, χ_{min}.

The length of the observation period and the value of ρ are determined empirically, as are k_{max}, $\chi_{initial}$ and δ_s/δ_c. Since each of these values must be determined as a function of the simulated system, the adaptive checkpointing approach suffers from the same drawback as does Lin's algorithm. Unless measurements are made prior to the use of these algorithms, neither can be optimal. On the other hand, it is important to keep the overhead engendered by these measurements as small as possible.

3 The Algorithm

As we mentioned in the introduction, we make use of Time Warp between clusters of LPs and a sequential algorithm within each cluster. A cluster is composed of one or more LPs, a Cluster Environment (CE), a Timezone table, and a Cluster Output Queue (COQ). The COQ holds copies of the messages that were sent by LPs in the cluster to LPs located in different clusters. This is necessary so that when the cluster receives either a straggler or an antimessage,

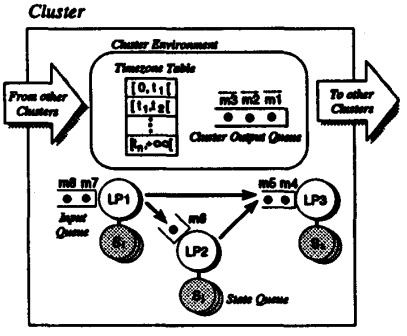


Figure 1: **Cluster Structure**

antimessages can be sent out to cancel incorrect computations. The CE is in charge of managing the COQ and the Timezone table (defined below) as well as the sending of antimessages. It is important to note that *individual* LPs do not send antimessages. On the whole, the cluster behaves like an LP in a purely optimistic system.

Figure 1 contains a cluster with 3 LPs. For each cluster, the simulation is decomposed into a series of non-overlapping time intervals we call *timezones*. When the simulation starts, each cluster has only one timezone, with interval $[0,+\infty[$. Each time a cluster receives a message from another cluster with timestamp t (the receive time of the message), it finds the timezone interval $[t_i,t_{i+1}[$ into which t fits and splits it into two new timezones with intervals $[t_i,t[$ and $[t,t_{i+1}[$. The message is then forwarded to the receiving LP.

Each LP maintains a Local Simulation Time (or LST). In addition, the LP keeps track of the Timestamp of the Last Event it processed (or TLE).

Before an LP processes an event, it first checks into which timezone the timestamp of the event fits. If that timezone is different from that of the last event which the LP processed, then the LP checkpoints by saving its state; otherwise the LP directly consumes the event. More generally, an LP checkpoints each time it changes timezones. When an LP sends a message to another LP located in the same cluster, it simply places it in the corresponding input queue of the destination LP. If the receiver is located in a different cluster, the sending LP passes the message to the Cluster Environment. The CE is then in charge of placing the message in the right cluster (containing the receiving LP) and keeping a copy of that message in the Cluster Output Queue. This is necessary in the case in which one or more messages have been sent within a timezone that has been invalidated. When this occurs, the CE sends an antimessage for each of these messages. This is achieved by checking whether or not the timezone of the messages in the COQ are still valid. If they are not, the antimessage is removed from the COQ and sent.

Suppose the cluster receives a straggler with timestamp t_s. As we have seen before, the CE creates a new timezone and it rolls back all LPs in the cluster which have a TLE greater than t_s to a checkpoint prior to t_s. In addition, the CE will send all the necessary antimessages stored in the COQ which have a send time greater than t_s. The cluster will proceed similarly when it receives an antimessage, with the difference that it will not create new timezones. After rolling back, the LP "coasts forward", as in Time Warp, not re-sending any messages produced before the time of the straggler. Also, the LP removes from its input queue all of the messages which have a send time greater than the timestamp of the straggler or of the antimessage which caused the rollback. This will not affect the correctness of the simulation as all the LPs in the cluster are rolled back. Hence all of the necessary messages will be regenerated. Since the events in the cluster are processed in strict timestamp order (i.e. lowest timestamp first), the descendants of the straggler will be placed correctly in the heap, and events at all of the LPs in the cluster will be processed in the correct order.

The LP is about to process event e.

```
    begin
(1)   find timezone Z_lp with interval [t_i, t_{i+1}[ s.t. TLE ∈ Z_lp
(2)   if t_r(e) ∉ Z_lp then checkpoint
(3)   TLE = t_r(e)
(4)   LST = max(LST, TLE)
(5)   simulate event e
(6)   LST = LST + service time
(7)   for all events e' to send do
(8)     t_s(e') = TLE
(9)     t_r(e') = LST
(10)    if destination LP of e' is in the same cluster then
(11)      insert e' into its input queue
        else
(12)      give e' to the CE for it to send
        endif
      endfor
    end.
```

The LP is told by the CE to roll back because the cluster has received an event.

```
    begin
                    /* Clean up the StateQueue */
(1)   for all states S ∈ State Queue s.t. TLE(S) > t_r(e) do
(2)     remove S from State Queue
      endfor
(3)   find state S ∈ State Queue
        s.t. TLE(S) ≯ TLE(S') ∀ S' ∈ State Queue
(4)   restore state S
                    /* Clean up the InputQueue */
(5)   for all events e' ∈ Input Queue s.t. t_s(e') > t_r(e)
(6)     remove e' from Input Queue
      endfor
                    /* Coast Forward */
(7)   while t_r(e') < t_r(e) where e' is the next event
(11)    TLE = t_r(e')
(12)    LST = max(LST, TLE)
(13)    simulate event e'
(14)    LST = LST + service time
(15)  endwhile
    end.
```

Figure 2: Pseudocode for the Logical Process

The cluster has received event e.

```
    begin
(1)   if e is not an antimessage then
(2)     find timezone Z with interval [t_i, ti + 1[ s.t. t_r(e) ∈ Z
(3)     create timezone Z_1 with interval [t_i, t_r(e)[
(4)     create timezone Z_2 with interval [t_r(e), t_{i+1}[
(5)     replace Z by Z_1 and Z_2
        endif
(6)   for all LPs in the cluster with a TLE ≥ t_r(e) do
(7)     tell LP to roll back to a state prior to t_r(e)
        endfor
(8)   for all antimessage e^{-1} ∈ COQ s.t. t_s(e^{-1}) ≥ t_r(e) do
(9)     send antimessage e^{-1} and remove it from the COQ
        endfor
    end.
```

An LP passes to the Cluster Environment event e to be sent.

```
    begin
(1)   send event e to the destination cluster
(2)   create e^{-1} where e^{-1} is the antimessage of e
(3)   insert e^{-1} into the COQ
    end.
```

Figure 3: Pseudocode for the Cluster Environment

The global virtual time (GVT) is the virtual time of the message or object which is the furthest behind in the system at a given time. It is necessary to compute a GVT estimate periodically in order to do garbage collection. We make use of a simple token-ring algorithm [2] as the number of processors which we make use of is small (less than 32).

Our fossil collection algorithm differs somewhat from that of Time Warp. In CTW, the state prior to the GVT must be saved, while in Time Warp this is not necessary. The reason for this is that it is possible to roll back prior to the GVT in CTW because not every state is checkpointed. Similarly, the events prior to the GVT in the LP input queue cannot all be removed since it is possible for the LP to rollback to a time prior to the GVT, since we might need to reprocess events with timestamps smaller than the GVT while coasting forward. Figures 2 and 3 contain pseudocode for the Logical Process and the Cluster Environment. We define $t_s(e)$ as the send time of event e and $t_r(e)$ as the receive time. We also define $TLE(S)$ as the TLE saved in state S.

3.1 Local vs. Clustered Rollback

In our algorithm, when a straggler or an antimessage arrives at the cluster, all of the LPs which have processed an event with a receive time larger than that of the straggler or of the antimessage will be rolled back.

This has the advantage of reducing memory consumption by discarding all of the messages in invalidated timezones (as they will be regenerated). However, the expense of forcing these LPs to roll back each time an antimessage or a straggler arrives at the cluster is not negligible, especially if most of the events generated by the LPs within that cluster are not causally related. In such a case, only a few of the LPs actually need to be rolled back. Hence a compromise was sought in which the decision of rolling back is made by the LP itself. In this new scheme, when a straggler or an antimessage is received, the cluster updates its timezone table accordingly and places the event into the input queue of the receiving LP. LPs now behave much like they do in a pure Time Warp system: rolling back when they detect the arrival of a straggler in their input queue and sending antimessages when needed.

Although this scheme might offer less overhead in terms of computation, it is more expensive in terms of memory since all the events in the LP input queues have to be kept (as they will not be regenerated).

We call the latter scheme *local rollback*, and the former scheme *clustered rollback*.

3.2 Local vs. Clustered Checkpointing

Another simple variant of Clustered Time-Warp was also designed in which an LP checkpoints only if it receives a message from an LP located in a different cluster (rather than checking whether it is entering a new timezone). This scheme is very simple to implement and requires less computational overhead than the previous schemes. Even though it is evident that an LP will have fewer checkpoints compared to the schemes described earlier, it is not obvious at all it will save more memory. On the contrary, and although it appears counter-intuitive, this scheme can be more greedy. Since the distance between checkpoints is greater, the number of events an LP needs to keep (in order to coast forward if it rolls back to a state prior to the GVT) tends to grow. Therefore, there is a trade-off: the fewer states an LP saves, the more events it needs to keep. In the case of logic simulation, the size of an event is far from being negligible compared to that of a state. Therefore the distance between checkpoints should not grow excessively if we want to keep the usage of the memory to a minimum.

We call this scheme *local checkpointing* as opposed to *clustered checkpointing* in which LPs save their state each time they enter a new timezone.

4 Simulation Model and Experiments

The implementation of the Logic Simulator was performed on a BBN Butterfly GP1000 shared-memory multiprocessor, and was written in C. Each node of the Butterfly has 4MB of local memory and a processor in the MC68000 family. The shared-memory is actually a virtually shared-memory.

We only made use of the shared-memory to implement a message passing system; therefore no global shared variable was used to implement any of the algorithms. This was done for two reasons: first, the results obtained from running the different algorithms will not be dependent on the presence of shared memory, hence making any comparisons unfair; second, porting the simulator to distributed memory machines such as the Intel Paragon will be more straightforward.

As we mentioned in the introduction, we oriented our algorithms towards the simulation of logic-level VLSI circuits. Our logic simulation model uses three discrete logic values: 1, 0 and undefined. To model the propagation delay, each gate has a constant service time. All of the common logical gates were implemented: AND, NAND, NOR, Flip-Flops, etc ...

The circuits used in our study are digital sequential circuits selected from the ISCAS'89 Benchmarks. Many circuits of different sizes have been tested, but we only present the results obtained from simulations of two of the largest circuits (table 1) since they are both representative of the results we obtained with the other circuits.

		inputs	outputs	flip-flops	number of gates
C1	s35932	35	320	1,728	18,148
C2	s38584	38	304	1,426	21,021

Table 1: Circuit C1 (s35932) and C2 (s38584)

A program was written to read the description file of the ISCAS benchmarks and to partition them into clusters. We used a string partitioning algorithm, because of its simplicity and especially because results have shown that it favors concurrency over cone partitioning; see for example [3].

A simulation run can be decomposed into 3 phases. First, each processor starts up by loading the gates assigned to it and by creating their corresponding LPs. Then, each gate which has an initialized state produces an event to broadcast its output to the gates connected to it. Some of these gates will be triggered and will propagate their changes throughout the circuit. After a while the system becomes stable, and events stop being generated. During the third phase, input vectors (previously randomly generated) are read and the simulation is run. Once the termination of the system is detected, statistics are collected.

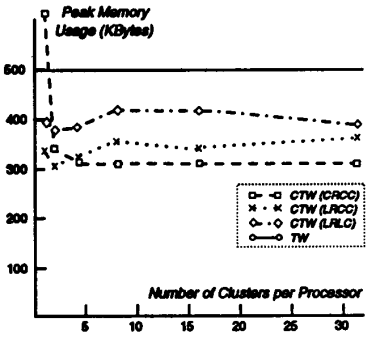
4.1 Experiments

We conducted two categories of experiments: one was to determine the effects of cluster size on the performance of each algorithm, and a second set of experiments to compare the performance (memory and execution time) of the algorithms with that of Time Warp. We used an aggressive cancellation mechanism.

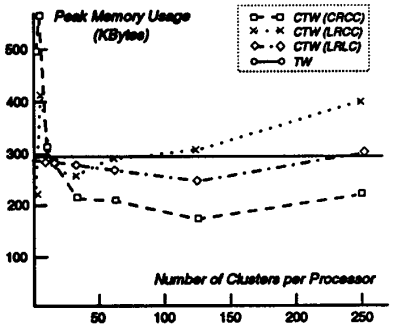
In the case of CRCC (Clustered Rollback, Clustered Checkpoint), larger clusters will result in more LPs being rolled back in the event that a straggler or an antimessage is received. For LRCC (Local Rollback, Clustered Checkpoint) and LRLC (Local Rollback, Local Checkpoint) smaller clusters result in a larger number of checkpoints and in greater memory usage.

In figures 4 and 5 we show the results obtained from simulating both circuits on 20 processors with different cluster sizes.

In figures 4a and 4b, we show the peak memory usage vs. the number of clusters per processor for C1 and C2 respectively. We define "peak memory usage" as the maximum amount of memory needed by any



a) Circuit C1



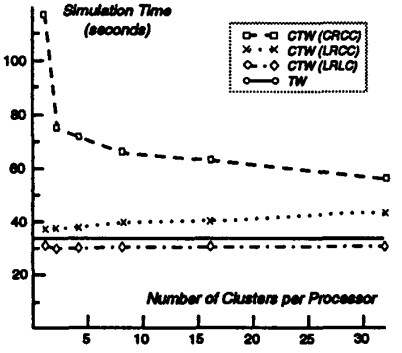
b) Circuit C2

Figure 4: Peak Memory vs. Number of Clusters per Processor

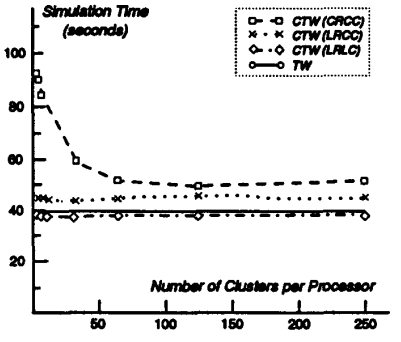
host during the entire simulation. It is dependent on the number of states and the number of events stored in memory (the ratio *state size/event size* was about 1.8 in our implementation). Both graphs indicate a rather stable behavior on the part of LRCC and LRLC with a minimal memory usage occurring at one cluster per processor. We also see up to a 40% difference in maximal memory usage between CRCC and Time Warp. CRCC, however, has a large maximal memory usage for one and two clusters per processor. However, the usage decreases dramatically, and is lower than either LRLC or LRCC from 4 clusters onwards.

We also observe a difference in the peak memory consumption between the two circuits for all of the algorithms. The reason for this difference between the two circuits is that C2 has an activity level nearly 3 times smaller than that of C1. Consequently, the calculated GVT tends to be closer to the actual GVT, and the fossil collection mechanism is then able to remove most of the useless states and events.

Figures 5a and 5b show the simulation time vs. the number of clusters per host. As we can see, CRCC has a significant overhead when compared to Time Warp. This is mainly due to the fact that some LPs are unnecessarily rolled back. Also, each time a clus-

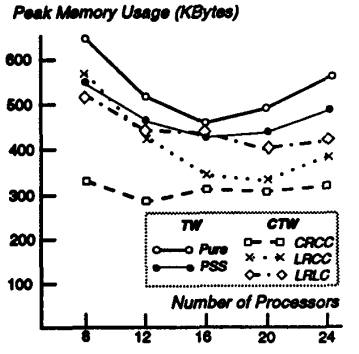


a) Circuit C1

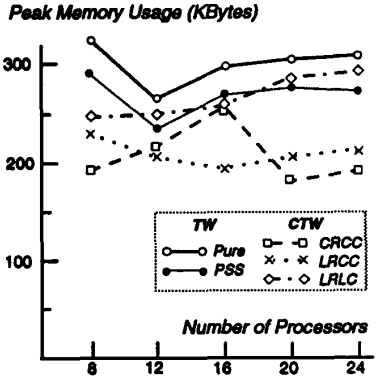


b) Circuit C2

Figure 5: Simulation Time vs. Number of Clusters per Processor



a) Circuit C1



b) Circuit C2

Figure 6: Peak Memory vs. Number of Processors

ter receives a straggler or an antimessage, the CE has to check all of its LPs to find out whether or not they have to be rolled back. This overhead becomes more pronounced when the cluster size is large. LRCC is a bit slower than pure Time-Warp since the cluster needs to update its timezone table regularly, and because LPs check the table each time they are about to process an event. As for LRLC, it is slightly faster than Time-Warp because fewer states are saved. In addition, the fossil collection mechanism has less work to do and can catch up quickly.

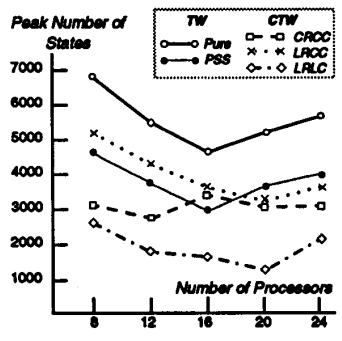
Based on these results, we chose the cluster size for each algorithm which gave the best performance in order to use them in our second set of experiments. For LRCC and LRLC, we chose one cluster per processor. In the case of CRCC, we chose 32 and 128 clusters per processor for C1 and C2.

In the second set of experiments we observed the behavior of the algorithms, varying the number of processors from 8 to 24. In figure 6, we show the peak memory usage of each algorithm vs. the number of processors for the circuits C1 and C2 respectively. We also show the performance of a Periodic State Saving mechanism which is a modified version of pure Time Warp in which the checkpoint interval is larger than one. We chose a checkpoint interval of 3 as it proved to be an optimal value for a large range of type of simulations [18]. In all cases, the proposed algorithms consume less memory than pure Time Warp, especially in the case of CRCC which made use of up to 50% less memory in circuit C1.

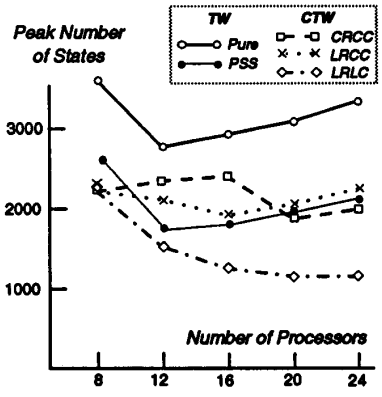
In figure 7 we see that all of the algorithms store far fewer states than Time Warp. In circuit C1, LRLC stores some 70% fewer states than pure Time Warp.

In figure 8, we present the simulation time of each algorithm vs. the number of processors. We observe that both LRCC and LRLC perform comparably to Time Warp. CRCC is from 30 to 60% slower than pure Time Warp in these examples. We note that this difference becomes less significant as the number of processors increase.

In table 2, we summarize the results by comparing each algorithm with pure Time Warp. For both circuits and for each algorithm, we give the minimum, the maximum and the average percentage difference from pure Time Warp for the maximum number of

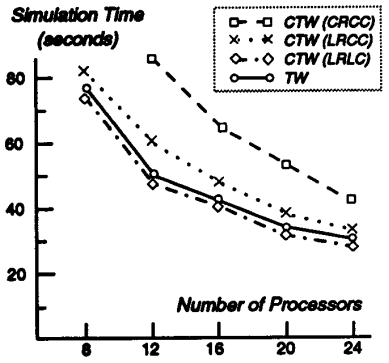


a) Circuit C1

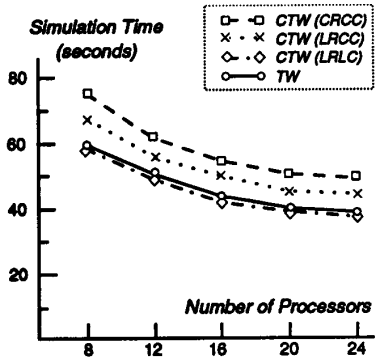


b) Circuit C2

Figure 7: Peak Number of States vs. Number of Processors



a) Circuit C1



b) Circuit C2

Figure 8: Simulation Time vs. Number of Processors

states, the peak usage of memory (in KBytes), and the simulation time (in seconds). We first observe that each algorithm saves a substantial number of states, especially for LRLC. However, these results do not necessarily directly translate into those obtained for total memory usage. Even though Periodic State Saving saves 30% of the states on the average (with a checkpoint interval of 3), the average memory saved does not go beyond 11%. This is because PSS needs to keep more events in order to restore LP states during the coast forward phase of rollback recovery. The same phenomenon is observed for LRLC. On the other hand, this does not happen when the Clustered Checkpointing mechanism is employed (ie: LRCC and CRCC), in which case, the performances are better in terms of memory consumption. These results underline the fact that simulation models such as Logic Simulation in which the size of the state of the LPs is not much larger than the size of the events, it is important to consider the increase of memory needed to store the supplementary events due to the checkpoint interval.

As to the simulation time, only CRCC is much slower than pure Time Warp, whereas the other algorithms exhibited a speed comparable to Time Warp. It should be noted that the current implementation of our simulator is not completely optimized (emphasis was put on the correctness of the simulator rather than its performance), so better results might be forthcoming for CRCC in the near future.

5 Conclusion

We have described in this paper the Clustered Time Warp algorithm (CTW), which makes use of Time Warp between clusters of LPs and a sequential algorithm within each cluster. Time Warp is (without regard to LP scheduling) traditionally implemented between the individual LPs of a model. It is our belief that CTW is useful when a model is comprised of a large number of LPs having low computational granularity, such as logic level VLSI models.

In this spirit, we presented three checkpointing al-

		States			Memory			Simulation Time		
		Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
C1	PSS	25	35	30	6	14	9	-2	0	0
	LRLC	60	73	65	5	25	16	2	6	4
	LRCC	20	34	27	12	32	22	-16	-6	-10
	CRCC	24	55	42	33	49	41	-75	-46	-61
C2	PSS	25	36	33	6	15	11	-2	0	-1
	LRLC	36	65	52	4	24	11	2	5	3
	LRCC	26	37	32	22	37	31	16	-10	-14
	CRCC	14	40	29	16	41	30	-29	-22	-26

Table 2: Performance Summary

gorithms for use with CTW and implemented them in a digital logic simulator. Two circuits were used: an 18,000 gate circuit and a 20,000 gate circuit. The 18,000 gate circuit had an activity level nearly three times that of the 20,000 gate circuit.

Each of the checkpointing algorithms represented a different memory vs. execution time trade-off. As we have seen in the preceding section, the CRCC algorithm saved between 35 and 50% of the maximal memory used in a Time Warp simulation. However, the price for this was a reduction in the speed of the algorithm from 30 to 60% as compared to Time Warp. The other two algorithms (LRLC and LRCC) decreased the maximal memory usage of Time Warp up to 30% without sacrificing much execution speed. Our results also pointed out a stable behavior of the algorithms with respect to the number of clusters employed. With this range of choices among checkpointing algorithms, it is possible to choose an algorithm depending upon the memory requirements of the simulation.

We believe that the clustering approach is useful for other purposes as well. One such example is dynamic load balancing, since instead of having to move individual LPs from one processor to another, clusters of LPs can be moved. Another point to note is that because message cancellations are performed at the cluster level, they tend to take place more quickly than if they are done on an LP level. Hence it is possible that CTW could avoid cascading rollbacks. We are presently at work in both of these areas.

References

[1] M.L. Bailey, J.V. Briner, R.D. Chamberlain, "Parallel Logic Simulation of VLSI Systems", ACM Computing Surveys, vol. 26, no.3,. September 1994, pp. 255-295

[2] S. Bellenot, "Global Virtual Time Algorithms", *PADS90*, pp. 122-127, vo. 22, no. 1, 1990

[3] J.V. Briner, Jr., "Fast Parallel Simulation of Digital Systems", *PADS91*, Anaheim, Calif., pp. 71-77.

[4] R.E. Bryant, Simulations of Packet Communication Architecture Computer Systems", T.R.-188, MIT LCSi, 1977

[5] K. Chandy, J. Misra, "Distributed Simulation:A Case Studey in Design and Verification of Distributed Programs", IEEE Trans. Software Eng., S-5, Sept. 1979

[6] K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via Sequence of Parallel Computations", *Comm. ACM*, Vol.24, No.11, pp.198–205, April 1981.

[7] R.M. Fujimoto, "Parallel Discrete Event Simulation", *CACM*, Vol.33, No.10, pp.31-53, 1990.

[8] N. Krivosidis, C.Tropper, "A Performance Analysis of Time Warp", *1992 Int. Conf, on parallel Proc.*, Chicago, Illinois

[9] D. Glazer and C. Tropper, "On local Balancing and Process Migration in Time Warp", *IEEE Trans. Parall. Dist. Comp.*, Vol. 4, No. 3 pp. 318–328, Mar. 1993.

[10] B. Groselj and C. Tropper, "A Deadlock Resolution Scheme for Distributed Simulation", *Proceedings of the SCS Eastern Multiconference* SCS Simulation Series, vol. 21, no. 2, pp. 108-112.

[11] B. Groselj and C. Tropper, "The distributed simulation of clustered processes", *Distributed Computing*, vol.4,pp. 111-121,1991.

[12] D.R. Jefferson, "Virtual Time", *ACM Trans. Prog. Lang. Syst.*, Vol. 7, No. 3, pp. 404–425, July 1985.

[13] Y. Lin, "Understanding the Limits of Optimistic and Conservative Parallel Simulation", Ph.D. thesis, Dept. of Computer Science and Engineering, University of Washington, TR 90-08-02, August 1990.

[14] Y. Lin et al, "Selecting the Checkpoint Interval in Time Warp Simulation", *PADS93* pp. 3-10, vol. 23, no. 1, May 1993.

[15] B. Lubachevsky, A. Schwartz, A. Weiss, "Rollback Sometimes Works ... if Filtered", Proc. 1989 Winter Simulation Conference, pp. 630-639, December 1989

[16] J. Misra, "Distributed Discrete–Event Simulation", *ACM Computing Surveys*, Vol.18, No.1, pp.39–65, Mar.1986.

[17] B. R. Preiss, I. MacIntyre, W. Loucks, "On the Trade-Off between Time and Space in Optimistic Parallel Discrete-Event Simulation", *PADS92*pp. 33-42.

[18] B. R. Preiss, W. Loucks, I. MacIntyre, "Effects of the Checkpoint Interval on Time and Space in Time Warp", ACM Transactions on Modeling and Computer Simulation, July 1994

[19] H. Rajaei, R. Ayani, L.E. Thorellu, "The Local Time Warp Approach to Parallel Simulation", *PADS93*

[20] R. Ronngren and R. Ayani, "Adaptive Checkpointing in Time Warp" *PADS94*, pp. 110-117.

[21] J. Steinman, "SPEEDES: A Unified Approach to Parallel Simulation", PADS92, SCS Simulation Series, vol. 24, no.3, pp. 75-84