The Dynamic Load Balancing of Clustered Time Warp for 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 dynamic load balancing algorithm developed for Clustered Time Warp, a hybrid approach which makes use of Time Warp between clusters of LPs and a sequential mechanism within the clusters. The load balancing algorithm focuses on distributing the load of the simulation evenly among the processors and then tries to reduce inter-processor communications. We make use of a triggering technique based on the throughput of the simulation system. The algorithm was implemented and its performance was measured using two of the largest benchmark digital circuits of the ISCAS'89 series. In order to measure the effects of the algorithm on workload distribution, inter-processor communication and rollbacks, we defined three distinct metrics.

Results show that by dynamically balancing the load, the throughput was improved by 40 to 100% when compared to Time Warp. Throughput is the number of *non rolled-back* message events per unit time.

When the algorithm tried to reduce inter-processor communication, rollbacks were substantially reduced. Nevertheless, no substantial improvement was observed on the overall simulation time, suggesting that load distribution is the most important factor to be taken into consideration in speeding up the simulation of digital circuits.

1 Introduction

Logic simulation poses a severe challenge to the PDES community due to the fine granularity of the computation, the very large number of basic elements, and the low level of circuit activity. Two different classes of algorithms are commonly employed to solve the synchronization problem of parallel simulation: the conservative approach introduced by Chandy, Misra [5], and Bryant [3], and the optimistic approach pioneered by Jefferson [9]. Conservative algorithms must either prevent or detect and break deadlocks. As for optimistic algorithms, processes must roll back to cancel *wrong* computations. Furthermore, memory management [6] and instability [11] are two fundamental problems of the Time Warp paradigm.

In an attempt to accommodate the low level of circuit activity, a hybrid approach, Clustered Time Warp (CTW) [1], was developed. CTW makes use of Time Warp between clusters of LPs and uses a sequential algorithm within the clusters. In CTW, several clusters can reside on the same processor, but a single cluster cannot be split among different processors. Three checkpointing algorithms were developed and represent a different memory vs. execution time trade-off.

Empirical results [13], have shown that a very strong locality exists in digital circuits, suggesting that historical information can be used to improve the mapping of the simulated model onto processors. In this spirit, we feel that the use of a dynamic load balancing technique can substantially improve the performance of logic simulations. Furthermore, the fact that CTW groups LPs into clusters makes the load balancing mechanism easier to implement since instead of having to deal with individual LPs, only clusters are considered.

Our dynamic load balancing algorithm attempts to evenly distribute the load among the processors. We believe the most important factor in load balancing of parallel VLSI simulations is keeping the processors as busy as possible, even at the expense of increased inter-processor communications and rollbacks. We provide evidence to support this contention later in the paper.

The remainder of the paper proceeds as follow. Section 2 discusses related results, while section 3 describes our algorithm in detail. Section 4 describes our experiments, and the concluding section follows.

2 Related results

In [12], Nicol and Reynolds present a statistical approach to dynamically partition a circuit and to map it to a set of processors. In their approach, a work graph is created to describe the precedence relations between the nodes. Edges in the work graph are weighted depending on the communication delays between the nodes and their overlap. If over a clock cycle, the time region of the activity of two nodes overlap, parallelism can be achieved by mapping these two nodes on two different processors. Given the work graph,

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

the authors partition the nodes into clusters by using a min-cut clustering algorithm based on Losen's approach [10]. The complexity of their algorithm is $O(E.(N-K).log_2(N-K))$ where N is the number of nodes in the work graph, E is the number of edges and K is the number of partitions. Their empirical results are positive. Nevertheless, the tests were done on a single 64-gate circuit and the authors assumed networks having no directed cycles. Because of the small size of the model, the lack of other tests, and the restrictions put on the connectivity graph of the circuit, it is extremely difficult to extrapolate the results to large sequential circuits.

Reiher and Jefferson introduce in [14] a new metric called the *effective utilization* which is "the proportion of work that is effective". The authors define effective work as the "work that will not be rolled back". Based on this metric, their load balancing algorithm moves logical processes from processors which are doing a lot of effective work to other processors which are doing little effective work. The performance results presented in the paper were obtained from running two typical benchmark simulations. One was a battlefield simulation, from which they obtained an improvement of 25% of the total simulation time by using their load balancing algorithm. The other model was a simulation of two-dimensional frictionless pucks moving and colliding on a table. Since the number of pucks used was quite high relatively to the number of processors, the simulation was naturally balanced and very little improvement was observed.

Burdorf and Marti [4] present a dynamic load balancing algorithm which executes on their Lisp-based Time Warp system running on a network of workstations. Their approach was motivated by the fact that users may load the workstations while a simulation was taking place, hence the need to move objects around to give the users a higher priority on computing resources. They chose the simulation time (Local Virtual Time) as a metric based on the assumption that rollbacks are extremely costly since they undo work which must be redone afterwards. Therefore, the main purpose of their algorithm is to reduce the variance between the objects' simulation times. By moving on the same processor objects which are far ahead in time and objects that are far behind, the authors believe that objects will synchronize with each others and less rollbacks will occur, hence speeding up the simulation. In their performance results, they find a five to ten times performance improvement over a simulation which does not use a dynamic load balancing strategy.

Glazer [7] [8] presents a dynamic load balancing strategy based on time slices. A time slice is a metric proportional to the ratio of the amount of computation time required by a process over its simulation advance. Once the time slice lengths are derived, processes are allocated to processors in an attempt to equalize the load on each processor. Three simulation models were constructed to represent different classes of simulation: a pipeline model, a hierarchical network model and a distributed network model. These models were ran on PARALLEX, a simulated multi-processor environment and the experimental results show that speedups ranging from 12% for the pipeline model up to 49% for the distributed network model were observed. Rollbacks were also decreased during the load balancing process, up to 50%.

In [15], the authors present a method for dynamic load-balancing for a simulator whose logical processes are grouped into clusters and which runs on a network of workstations. They introduce the Virtual Time Progress (VTP) which reflects how fast a simulation process continues in virtual time. Load imbalance is translated into a variation between the VTPs of the processors. By moving one or more clusters during the execution of the simulation, the load is balanced by trying to get all of the VTPs to be approximately the same value. Their results are quite encouraging since on a circuit of around 10,000 gates, they obtain a simulation runtime about 20% smaller than the time needed for the same simulation without load balancing. Nevertheless, only two workstations were used for the simulation and only one circuit was tested, so it is difficult to draw any general conclusion from these performance results.

A number of the existing load balancing strategies we have described above base their decision to invoke the load balancing algorithm on the progress of virtual time in real time. In the domain of logic simulation, the computational granularity is fine and is approximately the same at all of the LPs. Furthermore, the level of circuit activity is low. Consequently, we have decided to emphasize the role of the load in our dynamic load balancing algorithm and we have not used any virtual time metric.

3 The Algorithm

In this section, we describe our dynamic load balancing algorithm in detail.

3.1 Workload distribution

Due to the fine computational granularity of logic simulation, we need a metric to measure the load that is easy to compute and does not create too much overhead. We define the *load of a cluster* to be the number of events which were processed by its constituent logical processes since the last load balance in the simulation. This includes the rolled back events as well as the stragglers. Each processor also computes its load, which is the sum of the loads of all the clusters hosted by that processor. The load balance is improved by moving clusters from overloaded to underloaded processors. Given the load information of the clusters and the processors, our algorithm iteratively chooses the most loaded and the least loaded processors (respectively P_{heavy} and P_{light}). The load difference $\delta Load$ of both processors is then calculated. $\delta Load/2$ represents the load that must be transferred from P_{heavy} to P_{light} so that both are likely to have the same workload once the transfer has been performed. Since we want to move as few clusters as possible, we will choose the cluster whose load is the closest to $\delta Load/2$, and assign it to P_{light}. The load of P_{heavy} and P_{light} are then updated and the same procedure is executed iteratively.

In the current implementation of Clustered Time Warp, a processor called the *pilot* is dedicated to collecting statistics and other types of information from the processors involved in the simulation. In order to simplify the implementation, we assigned the load balancing task to this processor. Processors periodically send their load information to the *pilot* by piggybacking it on the GVT token.

3.2 Inter-Processor communication

Delays created by inter-processor communications may play an important role in determining the execution time of a parallel simulation. Consequently, we extend our algorithm to incorporate the communication factor. Instead of directly picking up the most heavily loaded cluster C_{heavy} in processor P_{heavy}, we will consider all the clusters whose load is close to that of C_{heavy}. We say that two clusters have approximately the same load when their difference is less than a certain tolerance. In our implementation, a tolerance of 10% was used. Then for each of these clusters, we evaluate the change that would occur in inter-processor communications if it is moved to any of the lightly loaded processors. The move that minimizes communication is then chosen.

Moving a cluster C_k from processor P_i to processor P_j is likely to alter the amount of communication between these two processors. It may worsen the situation since other clusters in P_i which are communicating with C_k will have to send events over the network. On the other hand, the situation is also improved since clusters in P_j which were communicating with C_k will not need to send messages over the network anymore. Therefore the overall change in communication load is:

$$\delta IPC(C_k, P_i, P_j) = \sum_{\forall C_n \in P_j} ICC(C_k, C_n) - \sum_{\forall C_m \in P_i} ICC(C_k, C_m)$$

where $ICC(C_a, C_b)$ is the number of messages exchanged between clusters C_a and C_b. The number of messages is calculated over a certain period of time which must be long enough so that the measure can be considered as reliable. In the case of logic simulation, this period of time must include the processing of at least one input vector so that all parts of the circuit have a chance to be activated. Since the load balancing algorithm is not activated until the whole system becomes stable, several input vectors which would have already been processed.

3.3 Pseudocode

Input: Π is the set of all processors.

Output: C_{move} is the cluster to move.
P_{dest} is the destination processor of C_{move}.

```
begin
(1)  P_dest ← ∅
(2)  C_move ← ∅
(3)  select P_light ∈ Π | Load(P_light) = Min_{P_i∈Π}(Load(P_i))
(4)  select P_heavy ∈ Π | Load(P_heavy) = Max_{P_i∈Π}(Load(P_i))
(5)  δLoad ← Load(P_heavy) − Load(P_light)
(6)  let Γ_c ⊂ P_heavy | ∀C_i ∈ Γ_c ⇒ Load(Ci) < δLoad/2
(7)  select C_heavy ∈ Γ_c | Load(C_heavy) = Max_{C_i∈Γ_c}(Load(C_i))
(8)  for each C_i ∈ Γ_c | Load(C_i) ≈ Load(C_heavy)
(9)    for each P_j ∈ Π | Load(P_heavy) − Load(P_j) > 2.Load(C_i)
(10)     if δIPC(C_i, P_heavy, P_j) < δIPC(C_move, P_heavy, P_dest)
(11)       P_dest ← P_j
(12)       C_move ← C_i
         endif
       endfor
     endfor
   end.
```

3.4 Triggering the load-balancing algorithm

Once a move has been decided upon, the loads of the two processors involved in the transfer are reevaluated. Then a decision process is started to find what cluster to move and where to move it. This procedure will converge to a better mapping of clusters, but not necessarily to the optimal one. This process can be repeated until the estimated workload distribution cannot be improved upon. Nevertheless, this procedure would not be very realistic for two main reasons. First, there is no control over the number of moves. Second, even though the processors' loads are reevaluated each time a move has been decided upon, the newly evaluated loads do not necessarily reflect the actual resulting load of the processors, mainly because the loads of the other processors have changed. Therefore, an iterative method was used. At each step of the load-balancing algorithm, only a certain number of clusters will be allowed to move. Then the system waits until the following two conditions are satisfied before triggering other moves:

- The cost of moving the clusters has been amortized by the resulting speed-up.
- New up-to-date measures are available.

Our load-balancing triggering mechanism is based on the *throughput* of the simulation system, defined as the number of *non rolled-back* message events per unit time. The throughput does not include anti-messages. In the domain of logic simulation, we feel that the throughput is a better measure of the overall speed of the simulation than the GVT advance. This is because the GVT advance is more dependant then the throughput on the nature of the model and its behavior. A large advance in GVT can be achieved by processing a small number of events.

Since the throughput fluctuates over time, a least-square approximation is used to obtain the general trend of throughputs, expressed by a first-degree equation. The system is considered stable when the coefficients obtained from the approximation do not vary by more then 5%. In our implementation, a throughput

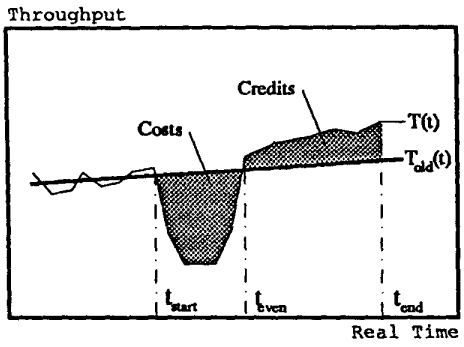

Figure 1: Throughput during a transfer

point is calculated every 3 seconds, the least-square approximation spans 6 points.

When a move is initiated at real time $t = t_{start}$, the throughput of the system tends to decrease since processors are spending their time transferring clusters. Then, once the operation is completed, the throughput increases, reaches its original value at $t = t_{even}$ which is equivalent to the break-even time (as defined in [15]). The throughput finally reaches a stable value larger than that previous to the transfer. Figure 1 depicts this situation.

Triggering the load balancing mechanism each time the stability is detected might actually decrease the overall speed of the simulation, even if the final throughput is higher than the previous one. As we have seen, moving clusters around has a cost in terms of throughput which is represented by the shaded area in the interval $[t_{start}, t_{even}]$ in figure 1. On the other hand, the transfer has the benefit of ultimately increasing the overall throughput of the system by an amount equal to the area above the interval $[t_{even}, t_{end}]$. If the next load balancing mechanism is triggered before the gain in throughput is equal to the cost of the transfer, the overall speed of the simulation might actually decrease. As a consequence, before launching the next load-balancing mechanism, we wait at least until the following condition is satisfied:

$$\int_{t_{start}}^{t_{end}} T(t) = \int_{t_{start}}^{t_{end}} T_{old}(t)$$

where $T_{old}(t)$ is the approximation of the throughput before the transfer and $T(t)$ is the actual throughput.

Observations have indicated that as the simulation progresses, the improvements become less significant and the period between each balancing adjustment grows longer. Once the cost of moving clusters does not improve the throughput so as to "pay back" the cost of the transfers, the load balancing mechanism is halted.

3.5 Metrics

To measure the effect of the load balancing mechanism, we define three metrics which depict different characteristics of the simulation system.

3.5.1 β: Workload distribution

To measure the quality of the load-balance, we define β as the ratio of the standard deviation of the processor loads to the maximum load observed. The lower that β is, the more equally the load is distributed.

$$\beta = \frac{1}{Load_{max}} \sqrt{\frac{\sum_{i=0}^{n-1} (Load(P_i) - \overline{Load})^2}{n-1}}$$

where $n = \mid \Pi \mid$ and Π is the set of all processors,

$$Load_{max} = Max(Load(P_i)) \; \forall P_i \in \Pi \text{ and}$$

$$\overline{Load} = \frac{\sum_{i=0}^{n-1} Load(P_i)}{n}$$

3.5.2 γ: Inter-processor communication

We define γ as being the ratio of the number of events exchanged between processors (C_{ipc}) to the number of events exchanged between clusters (C_{total}). The lower that γ is, the lower is the inter-processor communication.

$$\gamma = C_{ipc}/C_{total}$$

3.5.3 ρ: Cancelled computation

ρ is defined as the ratio of the number of events rolled back ($E_{cancelled}$) to the total number of events processed ($E_{processed}$). The lower that ρ is, the less computation is cancelled.

$$\rho = E_{cancelled}/E_{processed}$$

The three metrics β, γ, and ρ are measured over the same period of time as the throughput and only their mean calculated over the six previous points is considered.

4 Implementation and Experiments

4.1 Implementation

In Clustered Time Warp, each gate of a digital circuit is modelled by a Logical Process. LPs are then grouped into clusters which are in turn mapped onto processors. Several clusters may reside on the same node, but a single cluster cannot be split among different processors.

Three checkpointing algorithms were developed for CTW:

- **CRCC** Clustered Rollback, Clustered Checkpoint.
- **LRCC** Local Rollback, Clustered Checkpoint.
- **LRLC** Local Rollback, Local Checkpoint.

Each of these techniques offers a different memory vs. execution time trade-off [1]. CRCC is the least expensive in terms of memory, and LRLC is the least expensive in terms of time. In our experiments, we make exclusive use of LRCC since it offers an intermediate choice for both of these characteristics.

Cluster sizes in the range of 50 to 200 gates were experimented with. Since little difference was observed between these sizes, we present the results for 100 gates. We used a string partitioning algorithm, because of its simplicity and especially because results have shown that it favors concurrency over cone partitioning [2].

The dynamic load balancing algorithm was implemented on top of Clustered Time Warp and run on a BBN Butterfly GP1000 multiprocessor [1]. The implementation of message passing in the simulation is *independent of the shared memory of the Butterfly*. Consequently our results will apply to distributed memory architectures.

Moving a cluster from one processor to another is a 2-phase operation. First, the sending processor encodes the data structure of the cluster into a message and then sends it to the receiving processor. While the transfer is taking place, events are still sent to the original processor which stores them in a forward list. Once the transfer is over, the second phase of the transfer starts. The receiving processor sends an acknowledgment to the sending one which then sends it the forward list and broadcasts to all the other processors the new location of the cluster. Even though routing tables are updated immediately, due to variable communication delays it is still possible for a processor to receive messages for a cluster that has been moved away. In this case, the message is simply forwarded to the correct processor.

4.2 Experiments

We conducted a series of experiments in order to determine how well our dynamic load-balancing algorithm performs when compared to Time-Warp. We also tried to measure the effects of load distribution, interprocessor communication and rollbacks on the overall performance of the simulation. The circuits used in our study are digital circuits selected from the ISCAS'89 benchmarks. For the sake of the clarity, we only present the results obtained from simulations of two of the largest circuits (table 1) since they are both representative of the results which we obtained with other circuits and they have characteristics which result in two different behaviors.

First of all, we analyzed the effect of workload distribution without considering inter-processor communications and rollbacks. To this end, a series of simulations were ran on 20 processors, with pure Time Warp (TW), with Clustered Time Warp (CTW), and with CTW using our load balancing technique (BCTW).

		inputs	outputs	flip-flops	total
C1	s38417	28	106	1,636	23,949
C2	s38584	12	278	1,452	20,995

Table 1: Circuits C1 (s38417) and C2 (s38584)

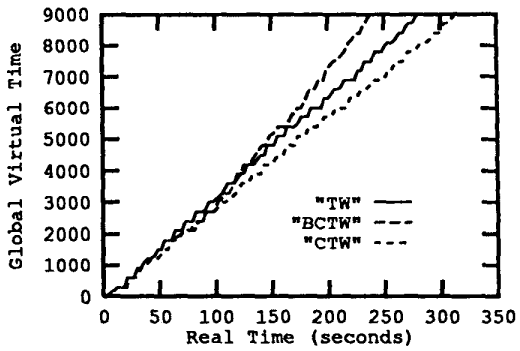

Figure 2: GVT Advance for C1

Each simulation consisted of the processing of about 500,000 events (cancelled events were not considered). Figures 2 and 3 show the progress of the Global Virtual Time versus the Real Time for C1 and C2 respectively, using TW, CTW and BCTW.

For both circuits, the figures clearly show that the total simulation time has been substantially decreased when load balancing was used. CTW is about 10% slower than pure TW for both circuits. This is due to the intrinsic properties of the LRCC algorithm used for CTW. LPs tend to roll back further in time when a straggler is detected since their checkpoint intervals tend to be larger than in Time Warp [1]. On the other hand, substantial memory savings are realized using

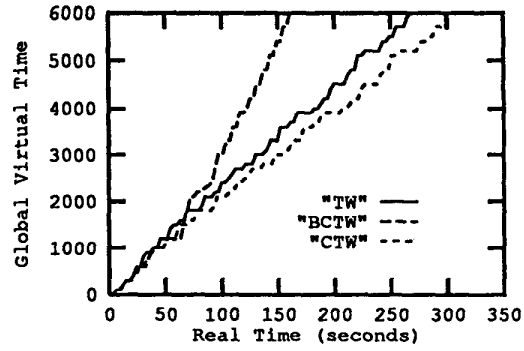

Figure 3: GVT Advance for C2

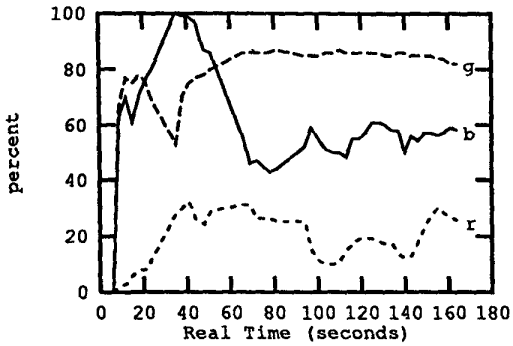

Figure 4: β, γ and ρ for C2

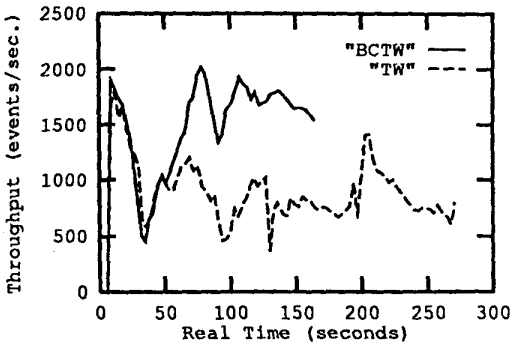

Figure 5: Throughput for C2

LRCC.

When using Clustered Time Warp with our load balancing algorithm, we observe for C1 a decrease of about 15% of the simulation time compared to Time Warp and 25% compared to CTW without load balancing. For C2, the simulation was 40% shorter compared to TW. The reason why load balancing had a more pronounced effect on circuit C2 may be found in the locality of its activity. The same partitioning and the same mapping algorithm were used for both circuits. Nevertheless, the workload distribution of C2 at the beginning of the simulation was not as good as that of C1, which is caused by a stronger locality of the activity in C2. As a consequence, moving clusters out of the overloaded processors and assigning them to underloaded processors tends to speed up the whole simulation more effectively for C2.

Figure 4 shows the variation of the three metrics β (load imbalance), γ (inter-processor communication) and ρ (rollbacks) during the simulation run of C2 using the Clustered Time Warp engine along with the dynamic load balancing mechanism. The first transfer of clusters was triggered 21 seconds after the beginning of the simulation. For a period of about 10 seconds, overloaded processors are busy sending the states of the clusters and logical processes which have been assigned to underloaded processors. During this time, fewer messages are processed and generated, which is shown by the sudden decrease of gamma. As a direct consequence, underloaded processors have less to do, and since overloaded processors are still busy, the load imbalance gets worse, explaining the increase of β. Once clusters of LPs have been transferred, β starts to decrease to a value lower than that at the beginning of the simulation, indicating that the load balance is better. But this improvement has a cost both in inter-processor communications and rollbacks. We can observe that about 86% of the messages exchanged between clusters are sent over the communication network, instead of 77% previous to transfer. This increase was expected since loaded clusters have been more evenly distributed over the processors. Rollbacks also increase from about 10% before the transfer to about 20 to 30% after the transfer. Nevertheless, despite the costs created by extra rollbacks and inter-processor communications, the overall throughput of the system increases. Even though the effect was less pronounced, we observed the same phenomenon for C1.

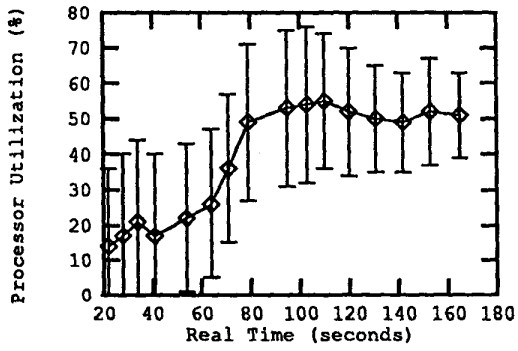

Figure 6: Mean and S.D. of the CPUs' activity for C2

Figure 5 shows the impact of the load-balancing algorithm on the throughput of the system during the simulation of C2. We can notice that soon after the transfer of clusters are initiated, the throughput becomes smaller than that of Time Warp. Once the transfers are over, it becomes twice as large as that of Time Warp.

For each processor, we measured the activity, which is the percentage of time spent on computation during a fixed period of time. At different points in the simulation, we calculated the mean and the standard deviation of the processor activities. The results are given in figure 6 and they show that as the load balancing mechanism transfers clusters from overloaded to underloaded processors, the activity rises from 25% up to 60%, and the standard deviation decreases by about 40%. This proves that the computation load became more evenly distributed over the processors.

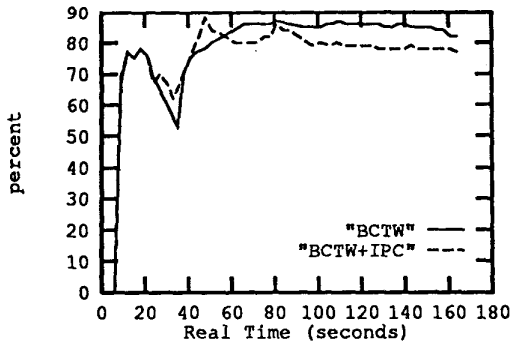

Figure 7: Effect on γ for C2 if IPC is minimized

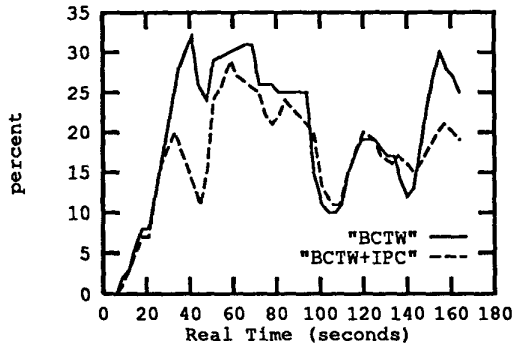

Figure 8: Effect on ρ for C2 if IPC is minimized

Similar results were obtained with C1, except for the fact that the throughput of the system did not increase as much as for C2. Instead of an increase of 100%, we observed an increase in the throughput of 40%. Note that C2 has a stronger locality of activity, which makes it easier for the load balancing algorithm to improve the overall throughput of the system.

If the load balancing mechanism tries to minimize inter-processor communications, we observe in figure 7 that γ could only be slightly decreased by about 5%. Figure 8 also shows that, on the average, the amount of computation cancelled by rollbacks has also been reduced, even though at some points, it is larger. Figure 9 shows for C2 the advance of the GVT versus real time for Clustered Time Warp with load balancing (BCTW), and Clustered Time Warp with load balancing considering inter-processor communication (BCTW+IPC). Even though we observe a small improvement of the total simulation time, it is obvious such an improvement is negligible when compared to that obtained with workload distribution alone.

Figure 10 shows the improvement of the throughput obtained by using the load balancing algorithm as

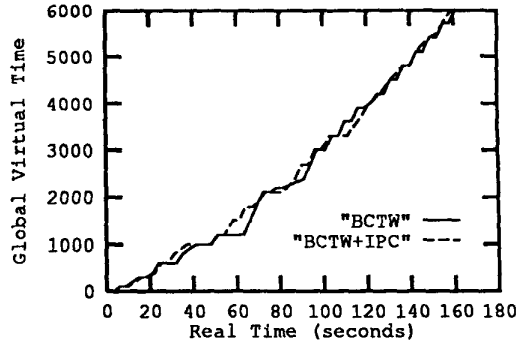

Figure 9: Effect on the GVT Advance for C2 if IPC is minimized

a function of the number of processors. We observe that as the number of processors increases, we obtain better performance. This is due to the fact that when a small number of processors are used, the load on the processors is much higher, thus leaving very little room to improve the load balancing. The improvement eventually levels off when the load is distributed among a large enough number of processors.

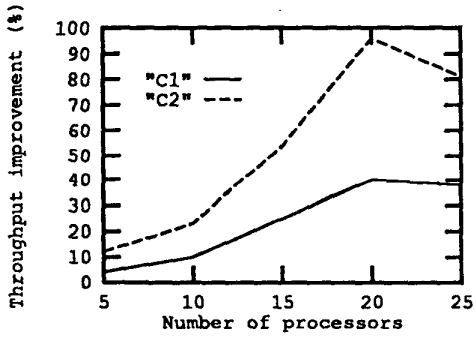

Figure 10: Improvement of the Throughput

5 Conclusion

We have described in this paper a dynamic load-balancing algorithm for Clustered Time Warp, a distributed logic simulator which makes use of Time Warp between clusters of LPs and a sequential algorithm within each cluster. The advantage of the clustering approach to load balancing is that instead of having to move individual LPs from one processor to another, clusters of LPs can be moved. We have also described a triggering technique based on the throughput of the simulation. Throughput is the number of

non rolled-back message events per unit time.

We have shown that a substantial acceleration of the simulation speed can be obtained. Two circuits of more than 20,000 gates were tested and improvement of 40 and 100% were obtained for the throughput.

Finally, we have shown that the improvement that can be obtained by reducing rollbacks and inter-processor communications is limited and that *the focus should be on evenly distributing the workload over the processors so as to keep them as busy as possible.*

References

[1] Hervé Avril, Carl Tropper, "Clustered Time Warp and Logic Simulation", pp112-119, PADS'95

[2] J.V. Briner Jr., "Fast Parallel Simulation of Digital Systems", PADS'91, pp71-77

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

[4] C. Burdorf, J. Marti, "Load Balancing Strategies for Time Warp on Multi-User Workstations", The Computer Journal, Vol.36, No.2, pp168-176, 1993

[5] K. Chandy, J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", IEEE Trans. Software Eng., S-5, pp440-453, Sept. 1979

[6] R.M. Fujimoto, "Parallel Discrete Event Simulation", CACM, Vol.33, No.10, pp31-53, 1990

[7] David M. Glazer, "Load Balancing Parallel Discrete Event Simulations", Ph.D. thesis, McGill University, 1993

[8] David M. Glazer, Carl Tropper, "A Dynamic Load Balancing Algorithm for Time Warp", pp318-327, Parallel and Distributed Systems, Vol.4, No.3, March 1993

[9] D.R. Jefferson, "Virtual Time", ACM Trans. Prog. Lang. Syst., Vol.7, No.3, pp404-425, July 1985

[10] S.L. Losen, "A Global Algorithm for the Multi-Partitioning of Graphs", M.Sc. thesis, University of Virginia, January 1985

[11] B. Lubachevsky, A. Schwartz, A. Weiss, "Rollback Sometimes Works... if Filtered", Proc.1989 Winter Simulation Conference, pp630-639, December 1989

[12] David M. Nicol, Paul F. Reynolds, Jr., "A Statistical Approach to Dynamic Partitioning", pp53-56, PADS'85

[13] B.L. Noble, R.D. Chamberlain, "Predicting the Future: Resource Requirements and Predictive Optimism", PADS'95, pp157-164

[14] Peter L. Reiher, David Jefferson, "Virtual Time Based Dynamic Load Management in the Time Warp Operating System", pp103-111, PADS'90

[15] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, H. Bauer, "Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations", pp175-180, PADS'95