

Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations

ROLF SCHLAGENHAFT AND MARTIN RUHWANDL AND
CHRISTIAN SPORRER

Institute of Electronic Design Automation
Department of Electrical Engineering
Technical University of Munich
80290 Munich, Germany
{ros,mir,chs}@regent.e-technik.tu-muenchen.de

HERBERT BAUER

Newbridge Networks
600 March Road
P.O. Box 13600
Kanata, Ontario
Canada K2K 2E6
hbauer@newbridge.com

Abstract

Performance of Time Warp simulation systems are often measured on exclusively available parallel computing resources. In distributed systems exclusive use is normally not feasible. Instead, due to the multi-tasking operating systems, many users share the workstations and their availability for parallel simulation purposes varies extensively. Time Warp has been found to be very sensitive to variations in available processing power. This paper presents two methods for a Time Warp VLSI simulation system to reduce the negative effect of a non-ideal environment on the execution of parallel simulations. A dynamic load balancing algorithm which adapts to the change of available processing power is presented. This mechanism, together with a multi-cluster partitioning technique significantly improves the performance of Time Warp based simulation systems on heterogeneous computing resources.

1 Introduction

Due to the increasing complexity of digital circuits, digital logic simulation suffers from consuming vast amounts of processing time and memory. Simulation time often exceeds days and memory requirements easily go beyond the capabilities of regular workstations. Parallel and distributed simulation methods have proven to be able to solve both problems. Time Warp [Jef85] is a well-known optimistic synchronization protocol for parallel simulations and achieves a significant reduction in elapsed time. Nevertheless, a detailed analysis of the obtained results shows, that there still is a strong potential for increasing the speed-up, since a large portion of time is spent for rollbacks. Therefore, a major goal of our research was to reduce the time needed for rollbacks.

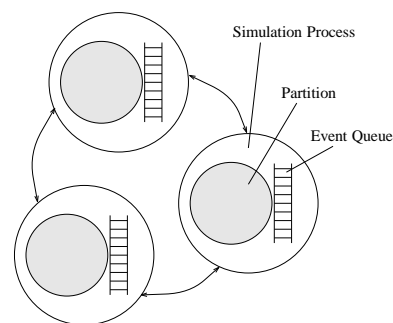


Figure 1: One partition per simulation process

In many simulation systems, each basic element is modelled as a single LP (logical process) [Fuj89, CGU⁺94, JB94]. This approach works very well for simulations of a fair number of basic elements (up to 1000), as is for example the case with telecommunication networks. However, in digital logic simulation, this is not feasible due to the large amount of gates (100,000 and more). Modelling each gate as a separate LP would result in high administrative overhead. Instead, our distributed simulator models a whole partition as one single LP, which can contain up to several thousand basic elements (see Figure 1). A drawback of this approach is, that whenever a rollback is necessary, the whole partition needs to be reevaluated, even if only a few elements are affected. The consequence is a waste of computing resources for redundant element calculations. In the following sections, we first present a method to reduce redundant element calculations and thereby increasing simulation speed. The basic idea is to divide and administer the circuit in more than one cluster per simulation process. With this multi-cluster administration the reevaluation after a rollback is limited to small regions without significantly

increasing the administrative overhead. Second, a dynamic load balancing mechanism is presented. Instead of using process migration techniques as found in distributed operating systems [Lud93, RJ90], we deploy a cluster migration method between simulation processes.

Our experiments show that both the number of rollbacks and the virtual time span rolled back is decreased by the improved load balance.

2 Multi-Cluster Simulator

Previous research [ML93] shows that Time Warp increases digital logic simulation speed in exclusively available computer networks. Now a new method is presented which decreases the time for a simulation run in a more realistic, loaded environment.

Let us consider a rollback on a simulator handling a single partition of the circuit. When it occurs, the whole partition is rolled back, i.e. all events evaluated in that partition after the rollback time are cancelled and simulation starts again at this time. When comparing the simulation results before the rollback and after the reevaluation, we find that very often results differ only in small regions of the partition. Obviously much time is spent on evaluating a large number of events a second time which often leads to the same simulation results. How can we prevent this?

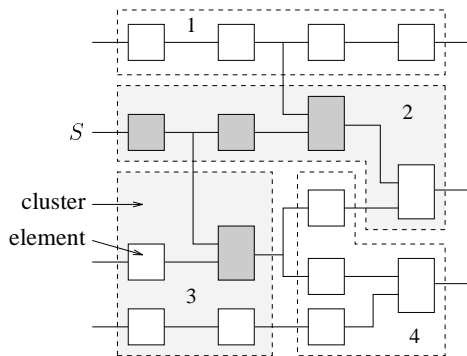


Figure 2: Multiple clustering

Only elements which are affected by the rollback should be recalculated. But the overhead for finding these elements is significant. A good compromise is to divide a partition in small clusters (100-1000 elements per cluster). Only a few of these clusters have to be rolled back and the rest of the partition remains unaffected. The example in Figure 2 lets us assume that the rollback caused by an event on signal S is executed with minimal costs when only the shaded elements are recalculated. Introducing clusters, the rollback can be restricted to the shaded clusters 2 and 3, i.e.

only the elements in these clusters have to be recalculated. So the execution costs of the rollback are closer to the minimal possible costs.

New Strategies: Introducing multiple clusters per simulation process leads to two new problems:

- An administration problem: How can a large number of clusters per simulation process be managed efficiently?
- A partitioning problem: How many clusters should be produced (What is the optimal size for one cluster?) and how should these clusters be mapped onto simulation processes?

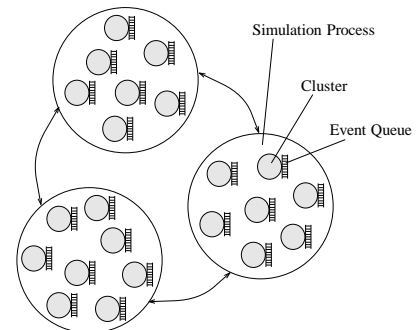


Figure 3: Multiple clusters per simulation process

2.1 Extended Administration

Global synchronization in our simulation system [BS93] is based on Jefferson's Virtual Time mechanism [Jef85], employing lazy cancellation and incremental state saving. The event queue of a simulation process is splitted into separate queues for each cluster (see Figure 3). They are scheduled in increasing time stamp order to guarantee causality. A time-wheel is used for fast cluster selection independent of the number of clusters. The administration overhead for the time-wheel is neglectable. It contains one entry per cluster holding the cluster identity and the next virtual time step, which has to be simulated for this cluster. The entry with the smallest virtual time stamp is dequeued first and one time step of the appropriate cluster is simulated.

2.2 Partitioning

Our simulation method requires a partitioning algorithm to divide a circuit into many small clusters and merge them into some larger sets, one for each simulation process. The Corolla approach [DBK90, SB93] yields good partitioning results for a large number of partitions. The basic idea is to detect strongly connected regions within a circuit and

to combine them into disjoint clusters called corollas by minimizing cut costs. The result is a fine grained clustering of the circuit with a large number of corollas. Then the corollas are merged to partitions in order to realize a good static load balancing and further reduce cut costs. Because of the large number of corollas, the partition size can be adjusted in a wide range. For the multi-cluster simulator the corolla approach can be extended in two ways:

- **Partition first:** First combine corollas to large partitions, one for each simulation process. Then combine the corollas of each partition to the number of clusters needed for each process.
- **Cluster first:** First combine corollas to the number of clusters needed for all simulation processes. Then combine clusters to partitions, which are assigned to the processes.

The partition first method results in a minimized number of cut signals between the simulation processes. With the cluster first strategy the cut signals between the clusters are minimized. Both partitioning methods were analyzed with different numbers of simulation processes (2-12) and clusters (2-300). We experience that if the number of all simulation events sent between simulation processes is less than 10% of all generated events, communication is no longer the bottleneck for speeding up simulation. For both partitioning strategies the number of cut signals between simulation processes is small enough for this condition to hold. But for dynamic load balancing, the second approach 'cluster first' is better suited, because cut costs between the different simulators remain small after moving a cluster (see section 3).

2.3 Results for Multi-Clustering

Multi-clustering reduces simulation time by restricting the effects of a rollback to clusters. The additional administration overhead consists of two things. A small constant expenditure must be spent for deciding, which cluster has to be simulated next. A more time consuming disadvantage is the overhead spent on administering the additional cut signals by extra data. In order to find out whether the improvements outweigh the introduced additional overhead, we ran simulations on two types of networks.

- An ideal, non-loaded network with Sun/Sparc2 workstations: The workstations are in a separate network trunk which is not used by other users.
- A non ideal, loaded network with DEC 5000/25 workstations: The workstations itself are non-loaded but on the network is normal traffic caused by other users and services.

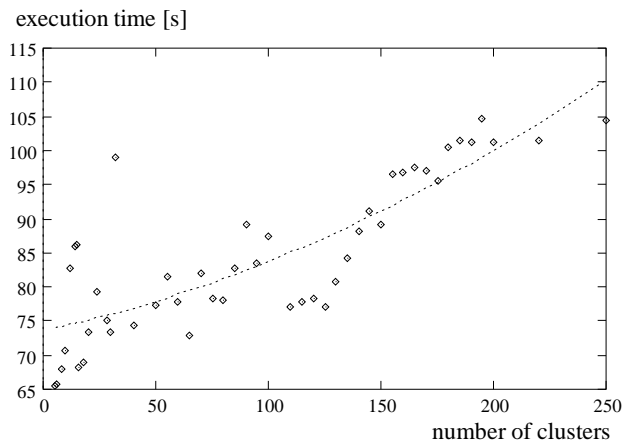


Figure 4: Five Workstations (Sun/Sparc2) with no network traffic

The results in Figure 4 and 5 are evaluated for the benchmark circuit s38584 [BBK89] with more than 20000 elements using five CPUs.

On the ideal network (Figure 4) there is no advantage with the multi-cluster method. In this situation, execution time slightly increases due to the additional administration overhead. In the network with regular net traffic (Figure 5) messages need more time to reach their destination. This results in more and 'longer' (in terms of virtual time) rollbacks. In this example, multi-clustering accelerates simulation up to 30%. In the shown example, simulation times employing 60 to 90 clusters (12-18 clusters per partition; 200-400 elements per cluster) are close to minimum. It should be noted that the multi-cluster simulator running with one partition per simulation process is only about 3 % slower when compared to the original parallel simulator, which can only manage one partition per simulation process. Due to the obtained results, the multi-cluster approach is a good base for dynamic load balancing.

3 Dynamic Load Balancing

3.1 Reasons

Very often Time Warp performance on a distributed network is measured in a perfect environment, where the available CPUs can be used exclusively. In such a situation load balancing of parallel logic simulation can be easily achieved by an appropriate partitioning at the start of the simulation as a preprocessing phase. This static partitioning can be used until the end of the simulation without modification and the number of rollbacks will remain small. In a realistic environment the situation is different. Nor-

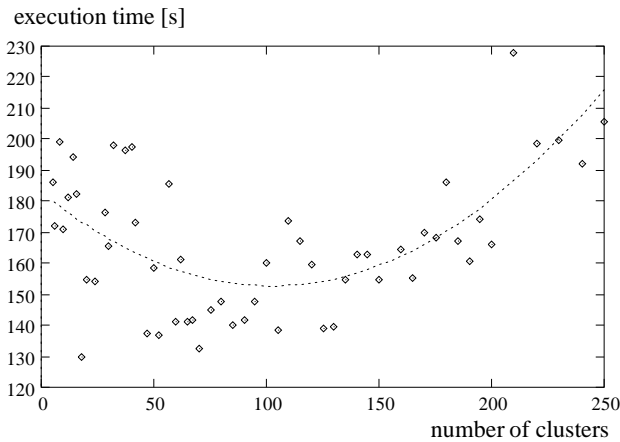


Figure 5: Five Workstations (DEC 5000/25) with normal network traffic

mally, users start jobs at arbitrary times on all available workstations. The computing power varies with time over the CPUs. Due to the optimistic synchronization protocol which relies on a more or less homogeneous progress of all simulation processes, the overall progress will be determined by the slowest CPU. The faster processes will be hit by many rollbacks and the available computing power is not used effectively any more [NPFR85].

If the unbalanced situation lasts for a certain period of time, one possibility is to redistribute the simulation ‘workload’ in order to speed up the overall progress. Our original parallel simulator is able to manage one partition on each simulation process, so there were three possibilities to do load balancing:

- Repartition the whole logic circuit
- Move single elements of the circuit
- Migrate processes

All three methods don’t work well for dynamic load balancing. In the first case, simulation must be synchronized, stopped and the repartitioning takes too much time. By the second approach a too small amount of work load is shifted and no significant change in the load situation is achieved. The third possibility depends strongly on the operating system and is not a general portable approach. This contradicts to one of our major goals: Doing the load balancing fully within the application program.

3.2 Feasibility of the Multi-Cluster Simulator for Load Balancing

To obtain a significant shift in workload, 5-10% of the circuit has to be transferred to another simulation process. In our example, this results in 200-400 elements. This can

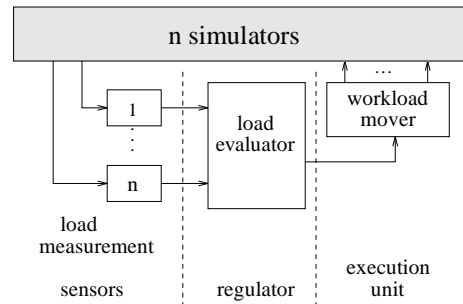


Figure 6: Control mechanism

be achieved by using the described multi-cluster simulator, since the size of clusters is adjustable before starting simulation by our partitioning tool. Moving a cluster to another simulator consist of the following steps: The static topology information of the cluster is made accessible to the destination simulator. Second, all signal and element states, pending events and information of incremental state saving concerning this cluster is moved. Last, all other simulators are informed about the change of the mapping. The advantages of introducing multi-cluster partitions remain and are not affected by transferring a cluster.

3.3 Controlling Load Balance

The load balancing problem can be described by means of ‘control engineering’. A regulator is needed which decides **when** to shift **which** cluster **from** which simulation process **to** which simulation process in an environment, whose load is changing stochastically. We solve this problem with an algorithm in the global control process, which consists of three main components (Figure 6) [Lud93]:

- load sensor
- load evaluator
- balancing adaptor

A more detailed view of the problems reveals that an appropriate measure for load and decision rules are needed which determine the necessary action in order to achieve load balancing.

3.4 Virtual Time Progress VTP

As an inverse measure for load, Virtual Time Progress (VTP) is introduced [Lim94]. It reflects, how fast a simulation process continues in virtual time. VTP is calculated as follows. First, we introduce the integrated virtual time (IVT) of a single simulation process with multiple clusters at a real time t_S , which is calculated at every simulation step S of any cluster on this process. ΔT_S is the length of simulation step S in virtual time units. n_S is the number

of clusters during that simulation step on this process. For IVT , we obtain:

$$IVT_{t_1} = \sum_{t_s \leq t_1} \left(\frac{\Delta T_s}{n_s} \right) \quad (1)$$

Time steps which are computed twice due to a rollback are included in this sum. During the interval $[t_1; t_2]$ the VTP of a single simulation process is defined as follows:

$$VTP_{[t_1; t_2]} = \frac{IVT_{t_2} - IVT_{t_1}}{t_2 - t_1} \quad (2)$$

An increase in the number of events to be simulated during a constant period of real time will result in a decrease of VTP , if the work load is unchanged. If in contrary the event count is unchanged but external load changes (processes launched etc.), VTP will decrease too. In both cases load balancing should be considered.

3.5 Break-Even Time t_{BE}

Of course, a move of a cluster needs a certain amount of time, which may not be neglected. During that period, the sending and receiving simulation processes cannot advance in virtual time. Therefore load balancing is only initiated if the ratio between the corresponding VTP s is big enough. Figure 7 shows how a break-even point can be derived. t_{rec} is the time needed to move a cluster, t_{BE} is the break-even time. The solid line (gradient VTP) is the measured advance of the global simulation progress (global virtual time) [BS92] before the load balancing. VTP_{pred} (dashed line) is the predicted VTP , if a load balancing is applied. VTP and VTP_{pred} can be easily calculated by the global control process, which has the information about the VTP s and number of clusters of all simulation processes. A load balancing is carried out, if the following condition holds:

$$VTP_{pred} > \frac{t_{BE_{max}}}{t_{BE_{max}} - t_{rec}} VTP \quad (3)$$

Our algorithm takes a maximum break even point as input. So the behaviour of the load balancing strategy can be adjusted to a certain load environment.

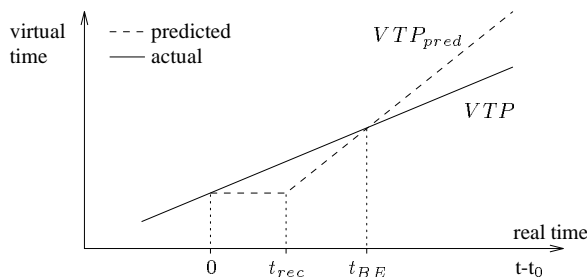


Figure 7: Break-even time

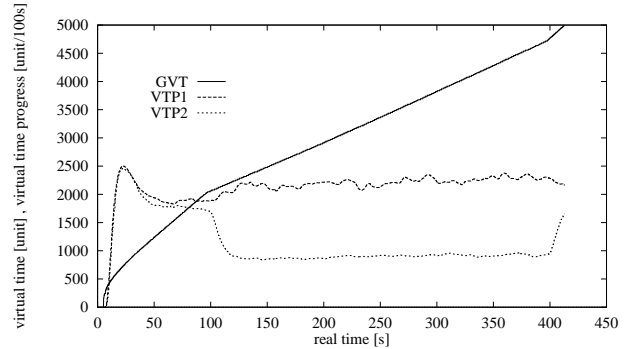


Figure 8: Without load balancing

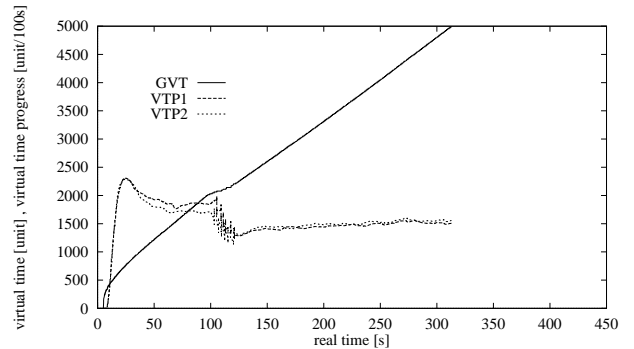


Figure 9: With load balancing

3.6 Results for Dynamic Load Balancing

Figure 8 shows a simulation run of the circuit s13207 [BBK89] on two processors without load balancing and an additional external load started at real time 100. Consequently, the VTP s largely differ. Further, GVT advances slower and the overall simulation progress is slowed down. Finally it ends at real time 420. In contrast, Figure 9 pictures the simulation of the same circuit in the same load environment but with load balancing. The first cluster is moved from the overloaded simulation process to the underloaded one only a short time after the VTP s begin to differ. Until real time 130 five more clusters follow (see Figure 10). Then the VTP s are adjusted again and the GVT advances faster than in the unbalanced simulation. In this situation, the simulation already terminates after 320 seconds. Figure 10 depicts a magnification of a part of Figure 9, where load balancing is carried out. The vertical bars in the plots of VTP_2 and VTP_1 represent the time points when the moved cluster is sent and received, respectively. After the transmission of a cluster, the slower simulation process (2) accelerates and process (1) is slowed down. The lag between two corresponding vertical bars shows the time needed for transmitting the cluster. It ranges between

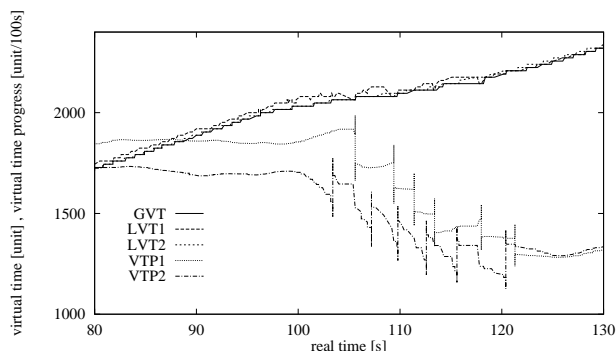


Figure 10: With load balancing - magnified

two and seven seconds, depending on the size of the cluster. The complete balancing process as a whole lasts 20 seconds in this case.

3.7 Conclusions and future work

It was shown that dynamic load balancing of a Time Warp simulation is feasible and yields good results. It can be achieved by slowly adapting the workload of the participating simulation processes to the load situation of the CPUs. Our plan for the future is to study the behaviour of our algorithm in a more complex environment. We want to extend our tests to cases with more than two CPUs. Further the reaction of our algorithm on faster and larger external load changes has to be investigated. The amount of data which has to be transferred between simulation processes to move a cluster is very large. Therefore, we want to apply data compression techniques on the moved clusters in order to reduce the transmission time t_{rec} .

4 Acknowledgement

The authors would like to thank Professor Dr. Kurt Antreich for his encouragement and valuable discussions. Further we want to thank Professor Dr. Brian Unger for the possibility to use the network of sun/sparc workstations at the University of Calgary. This work was supported by the German National Science Foundation (Deutsche Forschungsgemeinschaft) under grant SFB 0342.

References

- [BBK89] Franc Brglez, David Bryan, and Krzysztof Kozminski. Combinational profiles of sequential benchmark circuits. In *Proceedings IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 1929–1934, Portland, 1989.
- [BS92] Herbert Bauer and Christian Sporrer. Distributed logic simulation and an approach to asynchronous gvt-calculation. In

- Proceedings of the SCS Simulation Multiconference on Parallel and Distributed Simulation (PADS)*, pages 205–209, 1992.
- [BS93] Herbert Bauer and Christian Sporrer. Reducing rollback overhead in time-warp based distributed simulation with optimized incremental state saving. In *Proceedings Annual Simulation Symposium (ASS)*, 1993.
- [CGU⁺94] J. Cleary, F. Gomes, B. Unger, X. Zhonge, and R. Thudt. Cost of state saving and rollback. In *Proceedings Workshop on Parallel and Distributed Simulation (PADS)*, pages 94–101, 1994.
- [DBK90] Sujit Dey, Franc Brglez, and Gershon Kedem. Corolla based circuit partitioning and resynthesis. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 607–612, Orlando, 1990.
- [Fuj89] Richard Fujimoto. Time warp on a shared memory multiprocessor. *Transactions of the Society for Computer Simulation*, 6(3):211–239, July 1989.
- [JB94] Vikas Jha and Rajive L. Bagrodia. A unified framework for conservative and optimistic distributed simulation. In *Proceedings Workshop on Parallel and Distributed Simulation (PADS)*, pages 12–19, 1994.
- [Jef85] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.
- [Lim94] Walter Limmer. Verteilte Logiksimulation mit dynamischem Lastausgleich. Master's thesis, Lehrstuhl für Rechnergestütztes Entwerfen, Technische Universität München, 1994.
- [Lud93] Thomas Ludwig. *Automatische Lastverwaltung für Parallelrechner*. BI Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1993.
- [ML93] Naraig Manjikian and Wayne Louks. High performance parallel logic simulation on a network of workstations. In *Proceedings Workshop on Parallel and Distributed Simulation (PADS)*, pages 76–84, San Diego, 1993.
- [NPFR85] David M. Nicol and Jr. Paul F. Reynolds. A statistical approach to dynamic partitioning. In *Proceedings of the SCS Simulation Multiconference on Parallel and Distributed Simulation (PADS)*, pages 53–56, 1985.
- [RJ90] Peter L. Reiher and David Jefferson. Virtual time based dynamic load management in the time warp operating system. In *Proceedings of the SCS Simulation Multiconference on Parallel and Distributed Simulation (PADS)*, pages 103–111, 1990.
- [SB93] Christian Sporrer and Herbert Bauer. Corolla partitioning for distributed logic simulation of vlsi-circuits. In *Proceedings Workshop on Parallel and Distributed Simulation (PADS)*, pages 85–92, 1993.