VLSI Project

The focus of this research project is the distributed simulation of VLSI circuitry, making use of algorithms which have been developed by my research group at McGill.

Discrete-event simulation has been a major tool used to evaluate the performance of VLSI sytems and to uncover design faults. However, with the increase in the size and complexity of these systems we have been forced to confront the memory and speed limitations placed on us by a single processor. The use of distributed simulation, in which the simulator is executed as a distributed program over the nodes of a distributed machine, presents an attractive approach to this problem.

The fundamental problem posed by distributed simulation is determining how to synchronize the processes which comprise the program and which are resident on different the nodes of the machine.

Two major approaches have been developed for this problem, conservative and optimistic. In the conservative approach, an event is not executed at a simulation process until all of the events with smaller timestamps have been executed by the process. This means that it is certain that all of the events which can influence its outcome have already been executed. In the optimistic approach ,however, it is permissible to execute events without this certainty. If an event arrives which has a timestamp which is smaller than that of an already executed event, the associated process is rolled back to a safe state, and recommences execution. In order to obtain these states, it is necessary to periodically checkpoint the state of the processes. It is also necessary to eliminate the effects of messages which were incorrectly sent by a process. The optimistic approach attempts to take maximum advantage of the parallelism in the program.

I plan to make use of the Time of Next Event (TNE) algorithm and the Clustered Time Warp (CTW) algorithm, both developed by my research group in the course of this research. Our simulation platform will be a cluster of workstations on a high-speed network (Myrinet or Beowulf) because such a combination can offer true parallelism and can provide a cost-effective alternative to the use of supercomputers (e.g. Intel Paragon). Most of the research to date in parallel simulation has been conducted on large parallel machines.

TNE is a conservative algorithm which relies upon a shortest path algorithm which is independently executed in each processor in order to unblock LPs in the processor and to increase the parallelism of the simulation. It calculates a greatest lower bound on the arrival time of messages to all of the empty input queues in a processor in one invocation of the algorithm. Input to TNE consists of the LP graph, the minimum time stamps at all non-empty input queues nd the local simulation times at each LP. TNE is executed when the processor is idle, i.e. when no useful computation is being performed. It differs fundamentally from other conservative techniques which provide lookahead either by placing events into LP input queues or by using null messages. Because of a lack of global state information, inter-processor deadlocks can occur, and a deadlock detection/breaking algorithm must be employed. TNE was developed for a distributed memory environment. In a shared memory environment, it can be executed over the entire set of LPs, thereby avoiding inter -processor deadlocks. When compared to an optimized null message algorithm in the context of a queuing network simulation of a torus, TNE proved to be from 3-5 times as fast as the null message algorithm . A detailed performance analysis of TNE may also be found in. The shortest path algorithm was later extended to execute on collection of processors. The algorithm, the SemiGlobal TNE algorithm (SGTNE) executes a shortest path algorithm over a snapshot of all of the LPs in a cluster of processors. Partitioning algorithms based on the use of strongly connected components were also developed.

CTW is a hybrid algorithm in that it makes use of Time Warp between *clusters* of LPs and a

sequential algorithm within each cluster. CTW was developed with vlsi simulation in mind. We were trying to extract the maximum parallelism possible between clusters of LPs by using Time Warp while taking advantage of a sequential algorithm's modest memory requirements; Time Warp makes very large demands on memory in exchange for its speed. We developed 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 tested on the two largest circuits from the ISCAS89 benchmarks,(each had approximately 20,000 gates); their speed, the number of states saved and maximal memory consumption were compared to the same quantities for Time Warp. One of the algorithms saved an average of 40% of the maximal memory consumed by Time Warp, while the other two decreased the maximal usage by 15 and 22%, respectively. The latter two algorithms exhibited a speed comparable to Time Warp, while the first algorithm was 60% slower. A dynamic load balancing algorithm (token-based) was also developed for CTW. When tested on the two largest circuits in the ISCAS89 benchmarks, an improvement of 40-100% of the throughput was obtained. The stability as well as the scalability of CTW were investigated in . The motivation for this work was the possibility of a large memory consumption on the part of Time Warp as well as the fact that the number of rollbacks in a Time Warp simulation can increase without bound. Several large queuing network models (10,000-60,000 processes) were simulated. In all of the cases studied, CTW proved to be more scalable than Time Warp; in general the throughput of Time Warp decreases from 2 to 5 times faster than CTW. In addition, the memory requirements for CTW were between 2 and 3 times smaller than those of Time Warp. Our results strongly suggest that CTW has the capability to limit the explosion of rollbacks all too often associated with Time Warp.

We have also developed a flow control algorithm for Time Warp based on the use of stochastic learning automata and combined it with a dynamic load balancing algorithm. The algorithm is intended for use in a distributed memory environment as opposed to much of the work in memory managagement, which is oriented towards shared memory environments. Reductions in the execution time in the 20-30% range were obtained as well a large decrease in the number of rollbacks (50% in many cases).

We plan to compare the performance of both TNE and CTW on circuits which have been supplied to us by Nortel. Each of these approaches has potential pitfalls which need to be examined in the setting of large circuit simulations. TNE is subject to inter-processor deadlocks. While we managed to reduce the cost of deadlock detection, it still has the potential to cripple a simulation if called too often. At present, a heuristic constrains the frequency with which it is called. CTW, on the other hand, could suffer both from stability problems when executed on a sufficiently large circuit. While our experiments comparing CTW to Time Warp indicate that CTW is more scalable than Time Warp, they were conducted in a synthetic environment. Ultimately, we will be comparing the potential for speed inherent in the optimistic approach with the freedom from instability and potential for lower memory usage of the conservative approach.

Algorithms for circuit partitioning and for load balancing have established themselves as being of of fundamental importance in improving the the performance of circuit simulations. I have several projects which I wish to pursue in this area. One is to evaluating the utility of the integrated flow control and load balancing scheme for CTW which we alluded to above. It is oriented towards a distributed memory environment, and good results have been obtained in queuing network simulations.

A second project is to investigate whether a computationally inexpensive circuit partitioning algorithm can perform as well as a more expensive one when coupled with a dynamic load balancing algorithm (the one developed for CTW). An example of a computationally inexpensive partitioning algorithm is string partitioning. There is no lack of computationally expensive partitioning algorithms. Adaptive heuristic algorithms such as simulated annealing are very expensive, but can produce good results.(Simulated

annealing also requires an objective function to optimize). Probabilistic hill climbing and the sequence heuristic approach are variants of this approach. Graph theoretic approaches are also fall into the expensive category.

An interesting broadening of our vlsi research scope would be to parallel fault simulation. Little effort has been expanded in this important direction.