Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, eds.

ON THE SCALABILITY AND DYNAMIC LOAD BALANCING OF PARALLEL VERILOG SIMULATIONS

Sina Meraji
Wei Zhang
Carl Tropper

School of Computer Science
McGill University,
845 Sherbrooke St. W. Montreal,
Quebec, Canada H3A 2T5

ABSTRACT

As a consequence of Moore's law, the size of integrated circuits has grown extensively, resulting in simulation becoming the major bottleneck in the circuit design process. In this paper, we examine the performance of a parallel Verilog simulator on large, real designs. As previous work has made use of either relatively small benchmarks or synthetic circuits, the use of these circuits is far more realistic. We develop a parser for Verilog files enabling us to simulate in parallel all synthesizable Verilog circuits. We utilize four circuits as our test benches; the LEON Processor, the OpenSparc T2 processor and two Viterbi decoder circuits. We observed 4,000,000 events per second on 32 processors for the Viterbi decoder with 800k gates. A dynamic load balancing approach is also developed which uses a combination of centralized and distributed control in order to accommodate its use for large circuits.

1 INTRODUCTION

From Moore's law (Moore 2000) we know that the number of transistors which can be placed on an IC will double every 18 months. In order to design larger and more complex circuits, *Hardware Description Languages* (HDL) such as Verilog (Palnitkar 2003) and VHDL (Cohen 1995) are commonly employed. These languages are used for the formal description and simulation of modern digital circuits. The use of an HDL speeds up the design process and the time-to-market of these circuits.

A major part of the design process is verification. Verification engineers check the correctness and performance of the circuits using hardware and software simulation. In particular, for hardware simulation, it is hard to probe the values of internal signals and expensive to build complex simulators. As a consequence, the verification process relies on software simulation.

Sequential simulation can be employed as an accurate and inexpensive approach for the performance analysis of digital circuit. Unfortunately, the performance of a sequential simulation degrades when the size and complexity of the system being designed increases. Memory is an important issue. Current digital circuits have millions of gates and it is not easy to fit the simulation models of these circuits into a single processors' memory.

In addition to the demand for memory, the need for decreased simulation time is a major challenge for the verification process. Current *System-on-Chip* (SoC) circuits have CPUs, memory, I/O and other devices on a single chip. Hardware accelerators cannot keep up with this pace and as a result, the sequential simulation of digital circuits has become a bottleneck in the design process. In the meantime, parallel discrete event simulation has emerged as a viable alternative to provide a fast, cost effective approach for the performance analysis of complex systems.

A parallel (or distributed) simulation is composed of a set of sequential simulations which are executed on different processors and which model components of the physical system. Each of these sequential simulations is referred as a *Logical Process* (LP). Different LPs communicate with each other via time stamped messages. Events are stored in an *event list* and each LP processes all of its events in increasing time-stamp order.

Although processing the events in a sequential simulation is performed by using a centralized sorted list of events, this is not possible in parallel or distributed simulation. It is possible for events to arrive at an LP from other processors which, by their nature, have no access to the memory of other processors. Errors resulting from out of order execution

of events are referred as *causality errors*, and the problem of having each process execute events in time stamp order is referred as the *synchronization problem* (Fujimoto 1999). There are two main approaches for solving the synchronization problem: *conservative synchronization* (Lin, Fishwick, and Member 1996) and *optimistic synchronization* (Jefferson 1985). Conservative simulations rely on blocking, while optimistic simulations process events in the order in which they arrive at an LP, i.e. no attempt is made at assuring that events do not violate causality. The main drawback of conservative approaches is that they can not maximally exploit all of the available concurrency in a system. Among the optimistic synchronization schemes Jefferson's Time Warp (Jefferson 1985) is the first and most famous one. In this paper we use Time Warp for the parallel simulation of digital circuits.

Verilog and VHDL are the two major HDLs used for the design of integrated circuits. While some effort on distributed VHDL simulation had been made (Krishnaswamy and Banerjee 1995), (Lungeanu and Shi 2000), (Martin et al. 2002), this was not the case for Verilog. DVS (Li, Huang, and Tropper 2003) was the first environment for parallel Verilog simulation. Performance analysis of XTW (XU and Tropper 2005) has shown that it outperforms both Time Warp and Clustered Time Warp (Avril and Tropper 1995), which lay at the heart of DVS. However, XTW could not parse Verilog files. Hence we built a front-end for XTW rendering it capable of simulating any synthesizable Verilog design.

We show that the simulator's performance on large, real circuits is scalable. It is important to note that to date only small benchmark circuits and synthetic circuits were available for experimentation; our use of real designs in this paper provides more realistic benchmarks. It is also well known that balancing the load throughout the course of a simulation is of fundamental importance. Circuit simulation is no exception to this rule; large load imbalances were observed throughout the course of our experiments. Consequently, we developed a dynamic load balancing algorithm to deal with this problem. The algorithm emphasizes distributed control in order to deal with realistic circuits sizes. Previous algorithms made use of centralized control.

The rest of the paper is organized as follows. In section 2 we briefly describe Verilog and the structure of the XTW. In Section 3 we detail our efforts in adding the front end parser to XTW. Section 4 describes our dynamic load balancing algorithm. The performance analysis of four real circuits and dynamic load balancing algorithm are presented in section 5. Finally, the last section contains our conclusion and thoughts for future work.

2 BACKGROUND AND RELATED WORK

2.1 Verilog

The Verilog Hardware Description Language is an IEEE standard for designing Integrated Circuits (IC). Verilog can model both the behavioral and structural description of a circuit. An example of the behavioral and structural description of a one bit full adder from (Palnitkar 2003) is depicted in Figure 1. More information about Verilog can be found in (Palnitkar 2003).

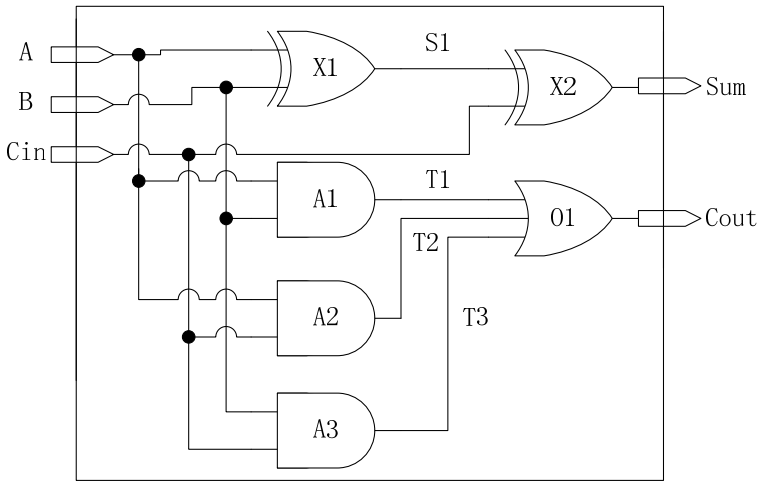
The basic element of the Verilog language is module which can contain the behavioural or structural code. Verilog models a digital circuit as a set of modules which are connected by input and output ports. Different connected modules implement various functional units of the circuit and Verilog allows the concurrent execution of these functional units. The available concurrency within the Verilog modules makes it an appropriate infrastructure for parallel simulation (Banerjee 1994).

2.2 XTW

In (XU and Tropper 2005), Xu and Tropper developed a new event scheduling and rollback mechanism, XEQ and rb-message respectively, which improve the performance of the optimistic logic simulation. XEQ has an O(1) cost, while rb-message eliminates the computing cost of anti-messages and also reduces the memory cost by eliminating the output queue in each LP. These two techniques are incorporated in a new simulator, referred to as XTW. XTW is an object oriented simulation environment which makes it an extendable environment for Time Warp. XTW utilizes the characteristics of digital circuits and makes the following simplifying assumptions:

- Events are generated in chronological order.
- An LP receives message in chronological order.
- LPs are sparsely connected.
- The topology of LPs is static during the simulation.

The first two assumptions lead to a zero cost for sorting events while the latter two assumptions make it feasible to implement XEQ and rb-messages in large scale simulations (XU and Tropper 2005).




```
module FA1(A, B, Cin, Sum, Cout);
  Input A, B, Cin;
  output Sum, Cout;
  reg Sum, Cout;
  reg T1, T2, T3;
  always @ (A or B or Cin)
  begin
    Sum = (A ^ B) ^ Cin;
    T1 = A & B;
    T2 = A & Cin;
    T3 = B & Cin;
    Cout = (T1| T2) | T3;
  end
endmodule
```

```
module FA1(A, B, Cin, Sum, Cout);
  input A, B, Cin ;
  output Sum, Cout;
  wire S1, T1, T2, T3;
  xor
    X1 (S1, A, B),
    X2 (Sum, S1, Cin);
  And
    A1 (T1, A, B),
    A2 (T2, A, Cin),
    A3 (T3, B, Cin);
  or
    O1 (Cout, T1, T2, T3);
endmodule
```

Figure 1: An example of Verilog structural and behavioural code

XTW employs clusters of LPs and the LRLC (Avril and Tropper 2001) technique from CTW (Avril and Tropper 1995). Each cluster has a multi-level event queue which is composed of three parts:

1. *Input channel* (InCh) which models a unique input of a circuit with the following rule:
   Rule 1: Each InCh can only have one unique incoming source. Each InCh itself contains one *input event queue* (ICEQ) and one *processed event queue* (ICPQ).
2. At the LP level, the event queue is referred as LPEQ, where events are sorted in increasing time stamp order.
3. At the cluster level, the event queue is referred as CLEQ where *time-buckets* are sorted in increasing time stamp order. A time-bucket is a set of events with equal time stamps.

There is a pointer at each input channel (referred as CIE) which points to the event which is de-queued from its ICEQ and is stored in the LPEQ or CLEQ. There is also a pointer at each LP (referred as CLE) which points to the event which is de-queued from its LPEQ and is stored in the CLEQ. These two pointers are used when rollback happens. According to (XU and Tropper 2005) XEQ has the following rules:

Rule 2: An InCh can submit one event to its corresponding LP's LPEQ if and only if ICEQ is not empty. The pointer value of the event is assigned to CIE.

Rule 3: An LP can submit only one event to its corresponding cluster's CLEQ if and only if LPEQ is not empty. The pointer value of the event is assigned to CLE.

The steps for processing an event in XTW are as follows:

1. After an event is generated it will be sent to the corresponding InCh and appended to its ICEQ.

2. If ICEQ is not empty, the smallest time stamp event is submitted to LPEQ according to rule 2 at a cost of 1.
3. The ICEQ event is inserted to LPEQ. The cost of finding the correct position is n where n, in the worst case, is the number of InChs at an LP.
4. If LPEQ is not empty, the smallest time stamp event will be submitted to the CLEQ according to rule 3. The cost of finding the correct position is m where m is equal to the number of LPs in the cluster in the worst case.

The consequence of the above four steps is that the cost of event scheduling is constant. Using rb-messages instead of anti-messages in XTW removes the need to employ an output queue. Whenever a straggler arrives, an rb-message is send to other LPs. This message will cancel all the messages which have a time stamp larger than the time stamp of itself.

3 DESIGN AND IMPLEMENTATION OF THE VERILOG PARSER

Digital systems are now described by Hardware Description Languages (HDL) because it is easier to design, read and synthesize HDL files (making use of Electronic Design Automation (EDA) tools). In (XU and Tropper 2005) the authors show that XTW has the best performance of the Time Warp based simulators which are employed for digital logic simulation. Unfortunately, XTW can only read bench files. In order to benefit from the performance of XTW, a front-end was added to it. This front-end changes the data format and generates the bench input data from the HDL descriptions. The Synopsys Design Compiler (DC) was used and a Verilog Parser was developed for the front-end.

3.1 Synopsis Design Compiler

The Synopsis Design Compiler (DC) can synthesize structural descriptions and behavioral descriptions of digital circuits into technology-dependent, gate-level designs and also do some optimizations for both combinational and sequential logics. A wide range of design formats including Verilog description, VHDL description and EDIF netlist files are supported by DC. Here we use DC to read the Verilog source files and to generate the corresponding gate level Verilog descriptions based on the generic technology library (GTECH).

The GTECH library is a standard cell library which contains over 100 basic modules. DC utilizes these basic modules in the GTECH library in order to describe the functionality of the original design. When these GTECH modules are created, a Verilog parser is utilized to convert them to a flat bench file which is readable by XTW.

3.2 Verilog Parser

The process of parsing consists of three steps: lexical analysis, syntax analysis and code generation. In the first step, we use LEX (Mason and Brown 1990) as a lexical analyser. LEX uses a table of regular expressions to recognize all of the words in the input file and to generate an output stream to save these words and their types. As an example, the word ”wire” would be assumed as the beginning of a wire-type variable statement and the word ”GTECH_AO21” would be assumed as the beginning of a basic GTECH_AO21’s instantiation gate.

After the lexical analysis, the words in the input file and their types are sent to a syntax analyzer and placed into different sentences. Finally, in the code generation step the corresponding code is produced from these GTECH gates according to the transformation rules. For example, a GTECH_AO21 gate would be replaced with an AND and OR gate.

3.3 Architecture

The main architecture of the current simulator is illustrated in Figure 2. It takes Verilog source file as the input and utilizes Synopsis DC to create GTECH modules. These modules are created using GTECH library. In the next step, the Verilog Parser parses these modules and creates the bench files. Verilog has a database of rules to create the bench files.

These bench files can be input to a distributed simulator (XTW in this paper). The simulator itself has 5 steps. In the first step LPs (gates) are distributed between different processors. Load-balancing, concurrency and communication cost are the most important factors for partitioning. Here we use a Depth First Search (DFS) partitioning scheme which assigns the same number of LPs to different processors.

In the next step, we initialize the primary inputs with a set of random input vectors. These events generate other events. The three remaining steps perform the simulation. The functions of different digital gates are implemented in the simulation executive. XTW is used as the Time Warp engine. Finally, the bottom layer is a communication layer which provides a

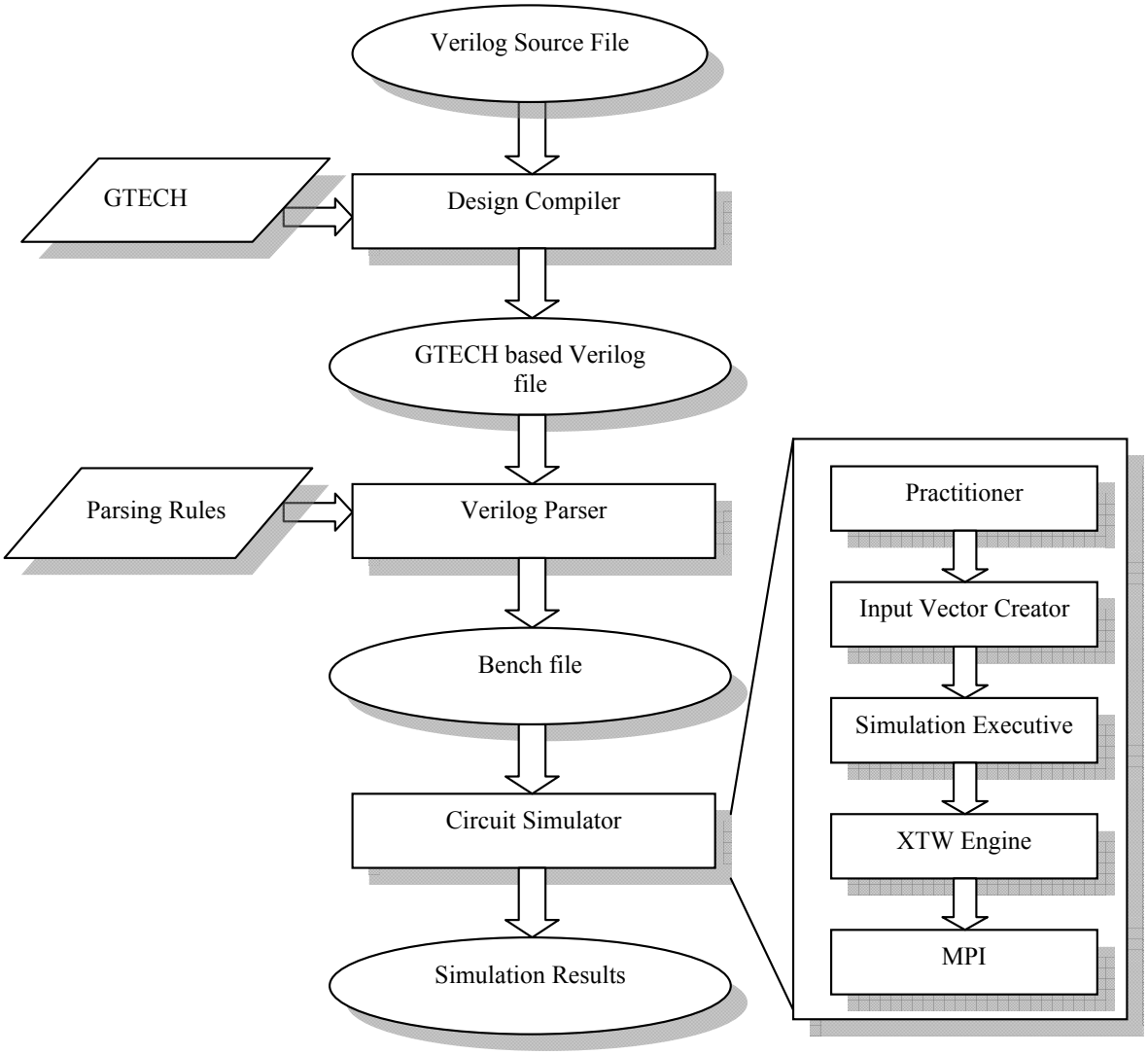


Figure 2: The main structure of the simulator

communication interface for the processors involved in the simulation. We use Message Passing Interface (MPI) (MPI 2009) for communication between different processors.

4 DYNAMIC LOAD BALANCING

In the course of our experiments we observed that the load on different processors in the simulation differed by as much as 12M events during the course of the simulation. As a result, we devised a dynamic load balancing algorithm to equalize the loads during the course of the simulation.

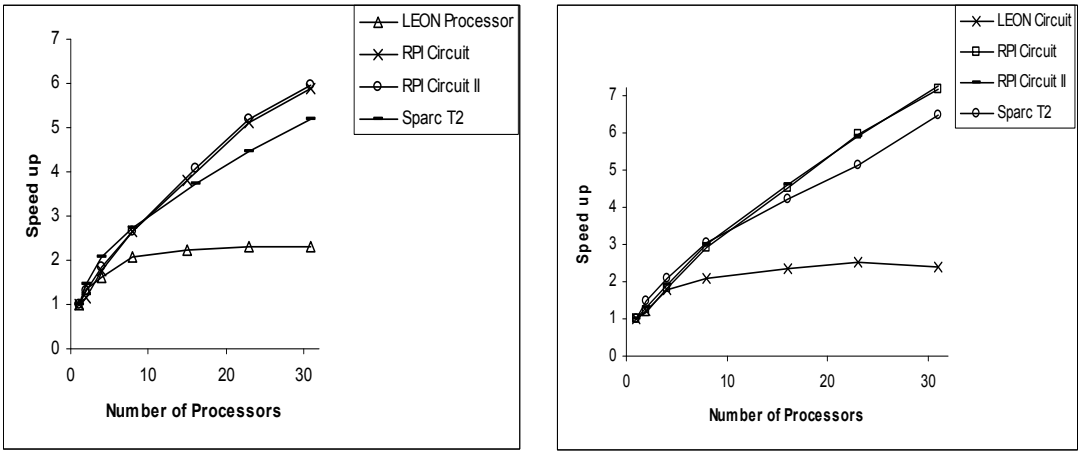
Dynamic load balancing for parallel circuit simulation studied in (Glazer and Tropper 1993, Schlagenhaft et al. 1995). However, the technique was applied to small circuits (up to 25k gates). Both (Schlagenhaft, Ruhwandl, Sporrer, and Bauer 1995, Avril and Tropper 1996) make use of a central node to select clusters of LPs and move them between processors. This was possible because the circuit sizes were small. The algorithm in (Avril and Tropper 1996) was implemented on a shared memory multi-processor and, as a result the communication cost of transferring the gates between computer nodes was negligible. This is not the case for current multi-computers. In addition, the benchmark circuits utilized by both of these authors had approximately 25,000 gates. As current circuits have millions of gates, we use a combination of centralized

and distributed control in our algorithm. In our approach, each processor sends the number of processed events (N) and the number of events it sent to other nodes since last receiving the latest GVT message to a central node. When the central node receives this data it selects the top P percent overloaded and under-loaded (computer) nodes and find the maximum bipartite matching between the under and over loaded nodes. For each pair the over-loaded node is informed about its corresponding under-loaded node. When an overloaded node receives such a notice, it selects up to L LPs which have the most communication with the corresponding under-loaded node and sends these LPs to the under-loaded node.

5 EXPERIMENTAL RESULTS

Our experimental platform consists of 32 dual core, 64 bit Intel processors. Each of these processors has 8 Gigabyte of internal memory. The distribution of the load between the two cores of a processor is automatically performed by the operating system. The processors are connected to each other by means of a 1 Gigabyte per second Ethernet. As previously mentioned, we employ Message Passing Interface (MPI) as the communication platform between processors. MPI provides a reliable mechanism for sending and receiving messages between different processors.

The Verilog source files used in this simulation are an OpenSPARC T2 (OpenSparc 2009) , the LEON processor (LEON 2009) and two Viterbi decoders designed at the Rennsalaer Polytechnic Institute (RPI). All of these circuits are open source designs. We used one core of the OpenSPARC T2 which is synthesizable by Synopsis DC and has 400k gates. LEON is a 32-bit microprocessor which is based on the SPARC-V8 RISC architecture and instruction set. It was originally designed by the European Space Research and Technology Centre, part of the European Space Agency, and after that by Gaisler Research (LEON 2009). One of the specifications of the LEON processor is its configurable core which makes it suitable for System-on-Chip (SOC) designs. The LEON processor has around 200k gates. The other circuit which used in our simulation are two Viterbi decoders with 100k and 800k gates from Rensselaer Polytechnic Institute (RPI).



(a) 10 input vectors

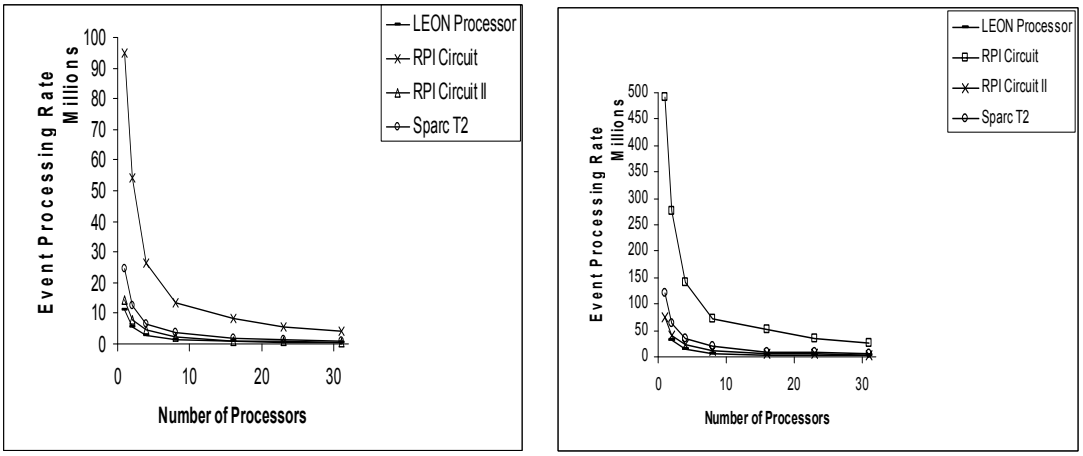
(b) 50 input vectors

Figure 3: Speedup vs. the number of processors

In our simulations we assume a unit delay for gates and zero transmission time for the wires. We employ DFS partitioning with load balancing constraint for distributing the LPs (gates) between different processors. In each simulation 10 or 50 random vectors are input to the circuit. Each point in our graphs is the average of 10 simulation runs.

Figure 3 shows the speedup vs. the number of processors for 10 and 50 random input vectors. As we can see, the speedup increases when the circuit is larger since more events are generated. When the number of processors is increased to 10 the speedup of the LEON Processor starts to flatten out. The reason for this lies both in the structure of the LEON processor and in its size. To begin with the Leon processor is more complicated than the flat RPI designs. resulting in a larger number of rollbacks. In addition, when the number of processors increases the communication cost increases and the speed of message cancellation during a rollback decreases. The results for the RPI circuits and the OpenSparc T2 show that the speedup increases when the number of processors and the size of the circuit increases. We note that the smaller RPI circuit has the same speedup as the larger one. The reason for this is that the RPI circuits are simple flat circuits which do not exhibit much feedback. Hence there is a lower probability of receiving out of order messages, resulting in fewer

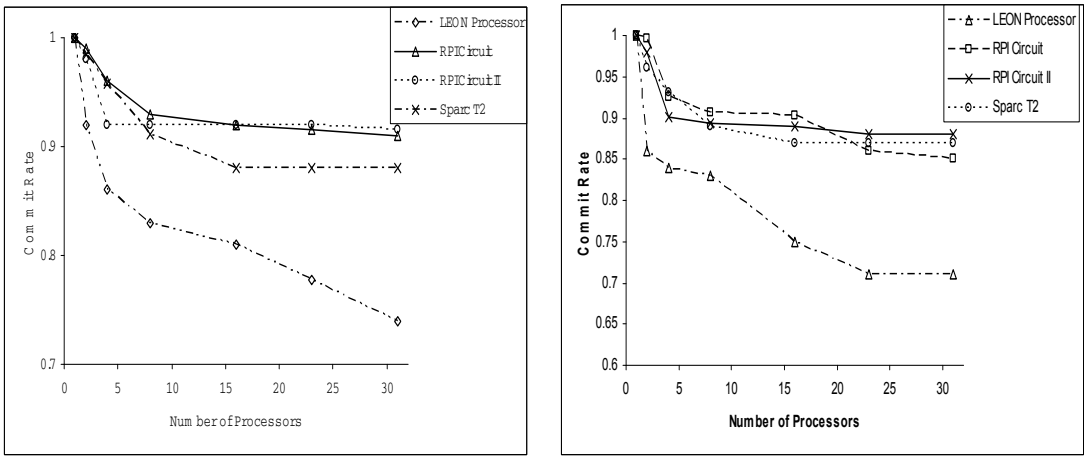
rollbacks and ultimately a better speedup. A comparison between the graphs of Figure 3 (a,b) also reveals that increasing the number of vectors results in better speedup. The reason for this is that the processors are kept busy and the processor idle time decreases. The maximum speed up for the larger RPI circuit with 10 and 50 input vectors were 5.9 and 7.16, respectively.



(a) 10 input vectors (b) 50 input vectors

Figure 4: Average number of processed events per processor

The total number of processed events per processor is shown in Figure 4 (a,b) for 10 and 50 input vectors, respectively. The total number of processed events was 500M for the larger RPI circuit when we have 50 input vectors. If all of the 32 processors are used, the event processing rates are 1.2M, 3M, 3.8M and 4M events per second for the LEON, OpenSparc T2, RPI (smaller) and RPI (larger) circuits respectively when the number of input vectors is 50. For 10 input vectors, the event processing rates are 1.1M, 2.1M, 2.8M and 4M events per second for LEON, OpenSparc T2, RPI (smaller) and RPI (larger) circuits respectively. Increasing the number of vectors results in bigger event rates because the CPUs have a larger processing load. For the larger RPI circuit we achieve the same event processing rate when we use 10 or 50 vectors. The reason for this is that we have large number of LPs and even with fewer input vectors the processors are already busy.



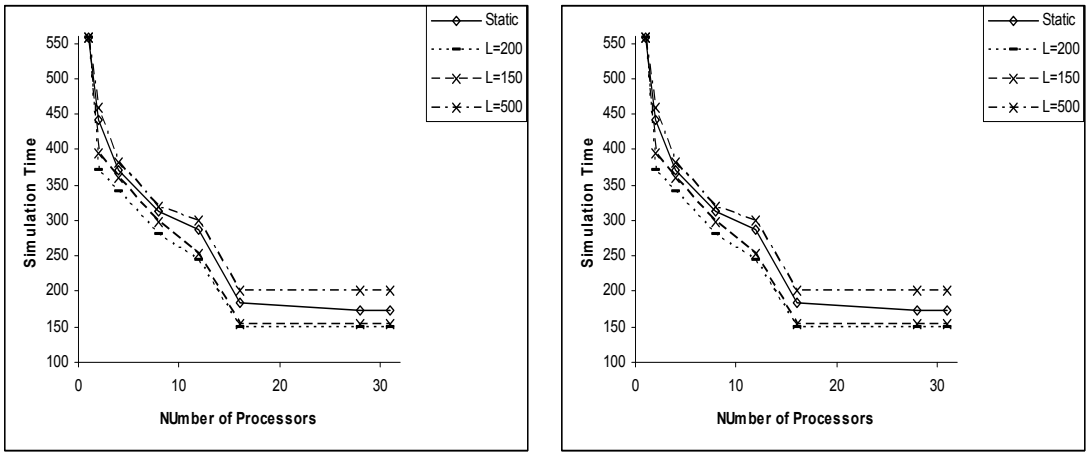
(a) 10 input vectors (b) 50 input vectors

Figure 5: Commit rate vs. different number of processors

Let us define the commit rate as the number of non-rollbacked messages divided by the total number of events. Figure 5(a, b) depicts the commit rate vs the number of processors for 10 and 50 input vectors. We note that the number of rollback messages increases with the number of processors. The reason for this is that the LPs are more spread out among the processors and as a result event cancellation takes longer. We can see that the LEON processor has the smallest commit

rate among the circuits. Once again, this is because of the structure and size of the circuit. More feedback results in a higher probability of receiving out of order events and, as a result the number of rollback messages increases and we need to send more anti-messages to cancel incorrect messages. The obvious effect of these rollback messages is the reduction of speedup as depicted in Figure 3. Increasing the number of vectors also results in smaller commit rates because more events are created resulting in a larger number of out of order events.

Figure 6 shows the performance of the dynamic load balancing algorithm on LEON processor for different values of P and L. P is the percentage of processors involved in the algorithm and L is the number of LPs that an over-loaded processor transfers in each round of the algorithm. We balanced the load up to 50% and achieved up to a 15% improvement in simulation time with P=20 and L=200. As can be seen, increasing the number of LPs from 150 to 200 results in better performance while an increase to 500 worsens the situation, a consequence of imposing too large a load on the communications system. The choice of P is also fundamental to the performance of the algorithm. The simulation time of the LEON processor is up to 7% better with P=20 than with P=10, also the result of an increased burden on the communications system. The above discussion shows that the values for P and L have a profound effect on the performance of the algorithm. Using a learning algorithm to find the best value of these parameters during the course of the simulation is absolutely necessary.



(a) P=10 (b) P=20

Figure 6: Simulation Time of the Dynamic Load Balancing Algorithm

6 CONCLUSIONS AND FUTURE WORK

In this paper, we made use of XTW as the simulation engine because it has exhibited the best performance among the Time Warp based circuit simulators. We used the Synopsis Design Compiler to generate GTECH modules from Verilog source files and developed a Verilog parser to convert the GTECH modules into a flattened bench file.

The performance of the simulator was evaluated with the LEON processor, the Open Sparc processor and with two Viterbi decoders designed at RPI. It is important to note that while previous work utilized small benchmark circuits and synthetic circuits, these circuits are real. We obtained an event rate of 4M events per second for the Viterbi decoder circuit on 32 processors. We noted that the speedup depended not just on the size of the circuit, but on its complexity as well-the flat RPI designs exhibited a better speedup then the more complicated open source designs. We developed a dynamic load balancing algorithm for parallel logic simulation which improves the simulation time up to 15%. The algorithm utilizes a combination of centralized and distributed approach for selecting the LPs which are transferred, an improvement upon the use of a purely centralized approach which were previously developed. The performance of the algorithm was shown to be critically affected by the number of LPs transferred and the number of processors involved in the transfer. Hence, developing a learning algorithm which can find good values for these parameters is certainly worthwhile.

REFERENCES

Avril, H., and C. Tropper. 1995. Clustered time warp and logic simulation. *SIGSIM Simul. Dig.* 25 (1): 112–119.

Avril, H., and C. Tropper. 1996. The dynamic load balancing of clustered time warp for logic simulation. *SIGSIM Simul. Dig.* 26 (1): 20–27.

Avril, H., and C. Tropper. 2001. on rolling back and checkpointing in time warp. *IEEE Transactions on Parallel and Distributed Systems* 12 (11): 1105–1121.

Banerjee, P. 1994. *Parallel algorithms for vlsi computer-aided design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Cohen, B. 1995. *Vhdl coding styles and methodologies*. Norwell, MA, USA: Kluwer Academic Publishers.

Fujimoto, R. M. 1999. *Parallel and distribution simulation systems*. New York, NY, USA: John Wiley & Sons, Inc.

Glazer, D. M., and C. Tropper. 1993. A dynamic load balancing algorithm for time warp. 318–327.

Jefferson, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7 (3): 404–425.

Krishnaswamy, V., and P. Banerjee. 1995. Design and implementation of an actor based parallel vhdl simulator. In *In 9th Workshop on parallel and distributed simulation(PADS95*, 135–143.

LEON Accessed on January 2009. *Open source processor*. http://www.gaisler.com/cms/.

Li, L., H. Huang, and C. Tropper. 2003. Dvs: An object-oriented framework for distributed verilog simulation. In *PADS '03: Proceedings of the seventeenth workshop on Parallel and distributed simulation*, 173. Washington, DC, USA: IEEE Computer Society.

Lin, Y.-B., P. A. Fishwick, and S. Member. 1996. Asynchronous parallel discrete event simulation. *IEEE Transactions on Systems, Man and Cybernetics* 26 (4): 397–412.

Lungeanu, D., and C.-J. R. Shi. 2000. Parallel and distributed vhdl simulation. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, 658–662. New York, NY, USA: ACM.

Martin, D. E., R. Radhakrishnan, D. M. Rao, M. Chetlur, K. Subramani, and P. A. Wilsey. 2002. Analysis and simulation of mixedtechnology vlsi systems. *Journal of Parallel and Distributed Computing* 2002:468–493.

Mason, T., and D. Brown. 1990. *Lex & yacc*. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

Moore, G. E. 2000. Cramming more components onto integrated circuits. 56–59.

MPI Accessed on January 2009. *Message passing interface*. http://www-unix.mcs.anl.gov/mpi/.

OpenSparc Accessed on January 2009. *Open source processor*. http://www.opensparc.net/.

Palnitkar, S. 2003. *Verilog®hdl: a guide to digital design and synthesis, second edition*. Upper Saddle River, NJ, USA: Prentice Hall Press.

Schlagenhaft, R., M. Ruhwandl, C. Sporrer, and H. Bauer. 1995. Dynamic load balancing of a multi-cluster simulator on a network of workstations. *SIGSIM Simul. Dig.* 25 (1): 175–180.

XU, Q., and C. Tropper. 2005. Xtw, a parallel and distributed logic simulator. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, 1064–1069. New York, NY, USA: ACM.

AUTHOR BIOGRAPHIES

SINA MERAJI is Sina Meraji is a PhD student in School of Computer Science of McGill University. His major research interest is parallel and distributed event simulation of Integrated circuits. He received his B.Sc. from computer engineering department of Amirkabir University, Iran. He also received his M.Sc. from computer engineering department of Sharif University, Iran. His email address is <smeraj@cs.mcgill.ca>.

WEI ZHANG Wei Zhang is currently visiting student in school of computer science of McGill University. he is also a Ph.D student in Computer Science, School of Computer Science, National University of Defense Technology, P.R. China. He received his B.Eng and M.Eng in Computer Science from School of Computer Science, National University of Defense Technology, P.R. China. His current research interests include Distributed Virtual Environments and Distributed Circuit Simulation. His email address is <weizhang@cs.mcgill.ca>.

CARL TROPPER is a Professor in the Department of Computer Science at McGill University. His major research interest is in parallel and distributed computing.He has worked in the area of distributed discrete event simulation since the inception of the field. His focus over the past several years has been parallel VLSI simulation. His group has developed a distributed VLSI simulation environment which is being used for research in both the synchronization and performance issues associated with VLSI simulation. Another research direction is the integration of parallel continuous and discrete event simulation models. His email address is <carl@cs.mcgill.ca>.