A Simulated Annealing Technique for Optimizing Time Warp Simulation

Wei Zhang[Ł], Sina Meraji[y], Jun Wang[y], Carl Tropper[y]
[Ł] *National Laboratory for Parallel and Distributed Processing, School of Computer Science*
National University of Defense Technology, ChangSha, 410073, China
[y] *School of Computer Science, McGill University, Montreal, H3A 2A7, Canada*
Email: weizhang@nudt.edu.cn, fsmeraj, jwang90, carlg@cs.mcgill.ca

***Abstract*—According to Moore's law the complexity of VLSI circuits has doubled approximately every two years, resulting in simulation becoming the major bottleneck in the circuit design process. Parallel and distributed simulations can be applied as fast, cost effective approaches to the simulation of large, complex circuits. In this paper, a simple yet effective simulated annealing-based approach is proposed to optimize the choice of a time window for optimistic parallel simulation. We chose gate level circuits simulations as our experimental vehicle. Our results show up to a 52% improvement in the simulation time using our simulated annealing algorithm. To the best of our knowledge, this is the first time that SA has been applied to optimize the performance of Time Warp simulations.**

***Keywords*-distributed simulation; time warp; simulated annealing;**

I. Introduction

Hardware Description Languages (HDL) such as Verilog[1] and VHDL[2] are commonly employed to design circuits. The use of an HDL expedites the design process and the time-to-market of these circuits. An important part of the design process is verification, in which engineers check the correctness and performance of the circuits using both hardware and software simulation. Because of the expense of hardware simulation, the verification process relies more upon software simulation. Because of the memory and speed limitation of sequential simulation, parallel discrete event simulation has emerged as a viable alternative to provide a fast, cost effective approach for the performance analysis of current VLSI circuits.

A parallel (or distributed) simulation is composed of a set of Logical Processes (LP) which are executed on different processors and which model different parts of a physical system. The events in a parallel simulation must be executed in the same order as they would be in a sequential simulation[3], i.e. causality must be maintained. In order to maintain causality, the LPs must be synchronized. There are two main approaches to the synchronization of a parallel simulation: conservative synchronization[4] and optimistic synchronization[5]. The main drawback of conservative approaches is that they cannot maximally exploit all of the available concurrency in a system. Worse yet, they can deadlock. Among the optimistic synchronization schemes, Time Warp[5] is widely employed. In this paper we use Time Warp for the parallel simulation of digital circuits. Time Warp simulators for digital logic circuits have been used in [6][7][8][9]. We also utilize DVS[6] for the parallel simulation of our circuits.

While Time Warp potentially makes better use of the system's parallelism, its over-optimism may lead to instability. In the worst case, the LPs spend most of their time rolling back in order to maintain causality, making it impossible for the simulation to progress[10]. Another major problem with Time Warp is its memory consumption, resulting from the need to save the states of LPs as well as events. One solution to this problem is to force an LP to block for a short period of time if its local time exceeds a certain virtual time (GVT). The virtual time constitutes the upper bound of a virtual time window, whose lower bound is the GVT. Approaches to computing the size of the time window may be found in [11][12][13]. Jun[13] achieved excellent performance by utilizing reinforcement learning approach[14].

Simulated Annealing (SA)[15] is a stochastic search method to find a good approximation to the global minimum of a function in a large search space. Due to the excellent performance of SA in solving combinatorial optimization problems, it has been applied in different applications, such as convex optimization[16], manufacturing cell formation[17] and the robust spanning tree problem[18]. In this paper we utilize an SA algorithm in order to find the size of a time window for Time Warp protocol. The simulation results show up to 30% improvement of simulation time in comparison to reinforcement learning[13].

The rest of the paper is organized as follows. In section 2, we briefly discuss our simulation environment. In section 3 we introduce the SA algorithm and how it is used in Time Warp. The performance analysis of this SA algorithm is addressed in section 4. Finally, the last section contains our conclusion and our thoughts for future work.

II. DVS: A Distributed Verilog Simulator

In [6], the authors developed a Distributed Verilog Simulator (DVS). DVS is an outgrowth of Clustered Time Warp (CTW)[7]. In CTW, as the name implies, LPs are grouped together in a cluster. The intuition behind this is that in large scale digital circuits (and network systems as well), LPs which model the same functional units should be grouped together to improve the performance of Time Warp. By keeping LPs which communicate frequently with

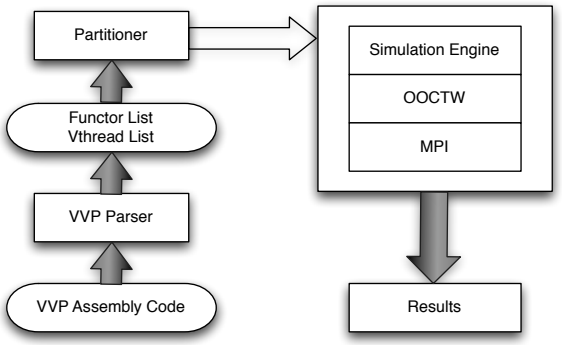

Figure 1. DVS Architecture

one another in the same cluster it is possible to reduce the amount of communication between the LPs. In CTW, Time Warp is used between different processors. Within each cluster a sequential algorithm is employed.

In DVS, Icarus Verilog is used to compile the Verilog source files to simulation codes. Icarus Verilog is an open-source Verilog simulator which is composed of two major parts: the Iverilog compiler and the Verilog Virtual Processor (VVP) simulator. These two are connected by the VVP assembly code. The Iverilog compiler translates the input Verilog file into the VVP assembly code. The VVP simulator, which is an event-driven simulator, processes the events and produces the final results. DVS receives the VVP assembly code generated by Iverilog as its input. As is well known, Verilog supports both the structural and behavioral description of a circuit. The structural description models a circuit as a network of interconnecting gates, and the behavioral description takes into account the changes in the signals. These two are translated to functor statements and thread statements in the VVP assembly code[6] .

The main structure of DVS is depicted in figure 1. The VVP Parser parses the VVP assembly code and instantiates structural and behavioral statements into Functors and Vthreads respectively. Each Functor is a digital gate with four inputs and one output. When any of the inputs changes during the simulation the value of the output port is updated. Vthreads are employed to drive functors with input vectors. The main function of the Partitioner in DVS is to provide an infrastructure for testing different partitioning algorithms. There is an abstract class, called ParBase, from which all of the partitioning algorithms can be derived.

There are three layers in DVS simulators. The bottom layer is a communication layer which is used by the upper layers for message passing. The middle layer is a distributed simulation layer, called OOCTW (Object Oriented Clustered Time Warp). This is an object oriented version of Clustered Time Warp which manages all of the Time Warp operations. The top layer is the simulation engine, which is a copy of the sequential DVS simulation engine. We added the SA algorithm to the Cluster class in the OOCTW layer of DVS.

III. Simulated Annealing

Simulated Annealing (SA) is a probabilistic heuristic approach originally proposed in [15] for global optimization problem of applied mathematics. For a given objective function f, SA can find a good approximation for the global optimum in a large search space. SA starts from an initial solution and attempts to move to a neighboring solution in order to improve the result. SA is grounded in an analogy pertaining to the thermal mobility of molecules, e.g. water changes from a liquid state to a pure crystalline form when the temperature is decreased.

A. The Simulated Annealing Algorithm

At each step of the algorithm, SA decides between moving the system to a new state s' and staying in the current state s where s' is a neighbor of the current state. This decision is made probabilistically. The probabilities are chosen so that SA tends to move to states with a better objective function value. States are, of course, specified in an application-specific way. This step is repeated until a "good enough" solution has been determined or a computational budget has been exhausted.

The acceptance probability function $P(\eth E; T)$ specifies the probability of making the transition from the current state s to a candidate new state s' according to changes in the value of the function $\eth E$. If a new state results in a better value, it is accepted. However, if the new solution yields a worse value, there is a probability that it will still be accepted. This is accomplished by first randomly selecting a number from (0,1). If the value is less than or equal to the value of the probability function, the new state is accepted; otherwise, it is rejected. This check prevents the algorithm from being stuck in a local minimum-a state that has a value which is larger than the global minimum, but which smaller than any of its neighbors. The probability function is shown in formula 1[15]:

$$P(\eth E; T) = e^{\eth E = T}; \quad (1)$$

where T is the current temperature. This is known as the Boltzmann distribution. When T goes to zero, the probability $P(\eth E; T)$ goes to zero if $\eth E > 0$, and to a positive value if $\eth E < 0$. On the other hand, for small values of T, the system will increasingly favor moves to state in which the objective function is improved and avoid those which worsen the objective function. When T is 0, the SA changes the state only if it improves the objective function.

Another feature of SA is the annealing schedule - the temperature is gradually reduced as the simulation proceeds. Initially, T is set to a high value in order to escape local minima. After a few steps the temperature T is reduced gradually according to some annealing schedule. The annealing schedule can be specified by the user but should end with a low value temperature by the end of the allotted computational budget. The evolution of the temperature T

during the optimization process is also called the cooling schedule.

B. Simulated Annealing and Optimizing Time Warp

Time Warp is prone to an explosive growth in the number of rollbacks and to an excessive usage of memory. One approach to avoiding these problems is to limit the optimism by allowing only those events whose timestamps are within a certain time window to be executed optimistically[12]. The time window is defined by the interval $[GVT, GVT + W]$, where W is the size of the window. Events within this interval can be executed, but those which have a timestamp beyond $GVT + W$ are not allowed to be executed, i.e. the LP is blocked. A blocked LP can still receive messages, but cannot send messages except for messages involved in the GVT computation. After a GVT update, the window itself is updated. Previously blocked LPs are unblocked if their next scheduled event falls within the newly updated window.

Note that if an objective function does not adequately reflect the main goal of the system, the SA algorithm may well fail to find an optimal policy. In Time Warp, the basic goal is to reduce the simulation time; hence the objective function should be related to the wall-clock time of the simulation. As in [13], we select the Event Commit Rate(ECR) as the objective function. If GVT_i is the wall clock time at the ith GVT, the Event Commit Rate (ECR) of the ith GVT interval (the interval from GVT_{i-1} to GVT_i) is defined as:

$$ECR_i = NC_i/(t_i - t_{i-1}); \quad (2)$$

where NC_i denotes the number of committed events at GVT_i.

Algorithm 1. *SA_initialize*()

```
1: old_wins[1::n] = random(1; MAX_WIN); %set initial windows randomly
2: bcast(old_wins); %broadcast the initial windows to all nodes
3: std_rate = 0;
```

Algorithm 2. *SA_doit*()

```
1: gather(total_ec); %gather committed events number
2: new_rate = total_ec/dt; %calculate the commited event rate
3: ž = e^((new_rate - std_rate)/T);
4: x = random(0;1);
5: if x < ž then
6:    old_wins = new_wins;
7:    std_rate = new_rate;
8:    new_wins = get_a_neighbor(old_wins);
9: else
10:   new_wins = get_a_neighbor(old_wins);
11: end if
12: bcast(new_wins); %broadcast the new windows
13: T_counter + +;
14: T = T_0 ð F_c^(T_counter=M);
```

The main process of our SA algorithm is described in algorithm 1 and algorithm 2. Since the result of the SA algorithm is independent of the initial choice, we set the window sizes of different nodes to a random value at the beginning of the simulation. Then, we use the first C GVT cycles to run the SA algorithm and find a good choice for the windows. After that, we will use the result of SA algorithm as the window sizes in the remaining simulation.

During the SA running period, we calculate the commit rate in every n GVT cycles, afterward we move to a neighboring window and try to find a better choice. In order to move to a new neighbor, we select some elements of the windows randomly and increase or decrease its size by a dynamic value, D, according to the acceptance rate. D is the distance between neighbors. It is an important parameter to adjust the SA process. We will show the effect of the distance in the next section. After finding a neighbor, we wait for n GVT cycles and calculate the new event commit rate ECR_{new}. If ECR_{new} is greater than old(previous) Event Commit Rate ECR_{old}, we accept the new neighbor as the current windows. Otherwise a random value between 0 and 1 is generated, if this random value is less than probability function value in (1), we again select the new neighbor as the current windows. As the cooling process, T is multiplied by a cooling factor, F_c, after each M GVT cycles, where F_c and M are some user input parameters.

IV. Experimental Results

In this section we present performance results for the simulated annealing algorithm. We utilized DVS[6] as our simulation engine and the C38417 and C38584 circuits from ISCAS-89 suite as our test benches. Our experimental platform consists of 12 dual core, 64 bit Intel processors. Each of these processors has 8 Gigabytes of internal memory. Load distribution 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. We utilized the Message Passing Interface (MPI) as the communication platform between processors. MPI provides a reliable mechanism for sending and receiving messages between different processors.

Among the different parameters which affect the performance of the SA algorithm, the distance parameter is very important. The distance parameter is the Euclidean distance between neighboring window-vectors. In order to find a neighbor window-vector of a current vector, we specify a radius and randomly select one within an n-dimension sphere. If the distance is too small, the searching process advances too slowly and we cannot find the optimal choice within a limited number of steps. If the distance is too big, we may jump too far and not obtain the optimal choice between neighbors.

Fig. 2 shows the effect of different distances on the performance of Time Warp for circuit C38584. In this figure, the X-axis represents the distance between neighbors and the Y-axis represents the simulation time. The numbers in the

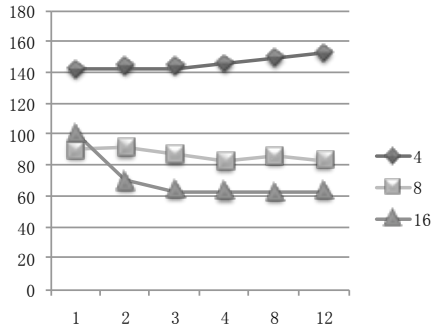

Figure 2. The effect of distance parameter on simulation time

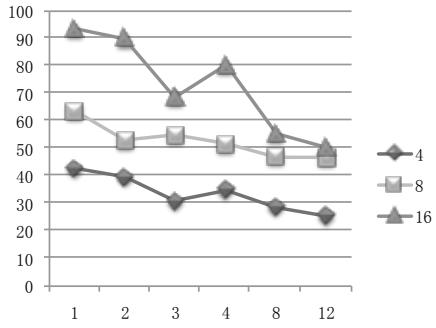

Figure 3. The effect of distance parameter on acceptance rate

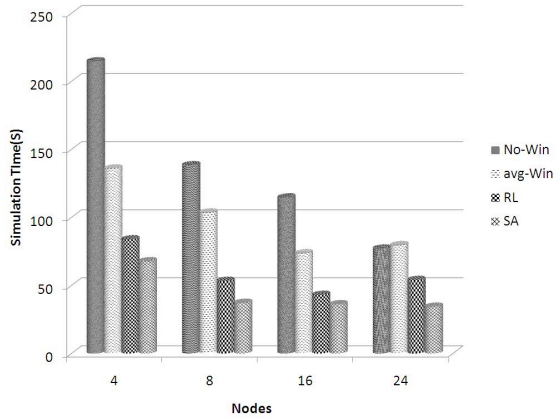

Figure 4. Simulation Results of Circuit C38417

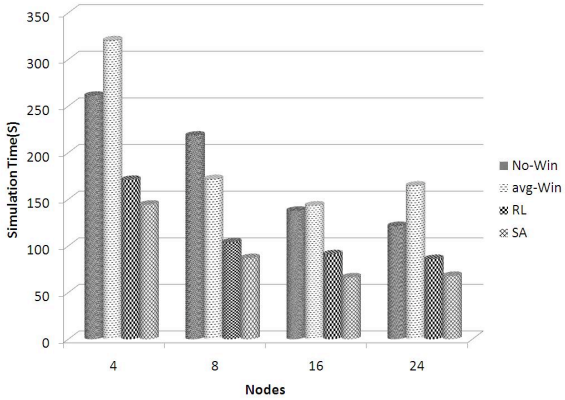

Figure 5. Simulation Results of Circuit C38584

legend represent the number of nodes which participate in the SA algorithm. As the results indicate, the best distances when we have 4, 8 and 16 nodes are 1, 4 and 8, respectively. The best distance becomes bigger with more nodes because the choice becomes greater with more nodes, and we need a larger distance to find a better choice.

There is a relationship between the simulation time and the acceptance rate. Fig. 3 shows the acceptance rates for different distance values. In this figure, the X-axis represents the distance between neighbors and the Y-axis represents the acceptance rate. The numbers in the legend represent the number of nodes which participate in the SA algorithm. As figures 2 and 3 imply, if the acceptance rate is close to 50%, the simulation time will be reduced by using the SA algorithm. The reason for this is that if the acceptance rate is too high, the neighboring window is too close to the original window and we need to increase the distance. On the other hand, if the acceptance rate is too low, the neighboring window is too far from the original window and we need to decrease the distance. Therefore, we applied an adaptive method to adjust the distance. If the acceptance rate is greater than 50%, we increase the distance; otherwise we decrease it.

Figures 4 and 5 show the simulation results for the C38417 and C38584 circuits with different algorithms. In these figures the X-axis represents the number of nodes that participate in the simulation and the Y-axis represents the simulation time. The labels in the legend represent different methods which we utilize in order to find the size of the time window. "No-Win" represents the results without using a time window (the window is set to infinity); "avg-Win" represents the average results for all of the values of time windows; "RL" represents the results of a Reinforcement Learning algorithm with random candidates, in this work we implement a Q-Learning algorithm with one state; "SA" represents our simulated annealing algorithm. Both "RL" and "SA" methods are implemented on-line. As these figures indicate, the SA algorithm outperforms the other algorithms. The results show that simulation time is reduced by up to 52% when compared to the "No-Win" approach for the C38417 circuit. The best simulation time in [13] was achieved by applying the reinforcement learning approach. As we can see, our SA algorithm reduced the simulation time by up to 36% compared to the approach in [13] for the C38417 circuit with 24 nodes.

Figures 6 and 7 show the standard variance for the experiments for 100 repetitions with different methods. In the figures, the X-axis represents the number of nodes and the Y-axis represents the standard variance. The labels in the legend represent the different methods. From these figures, we can see that our SA algorithm gets more stable result than the RL algorithm. The reason for this is that the result of

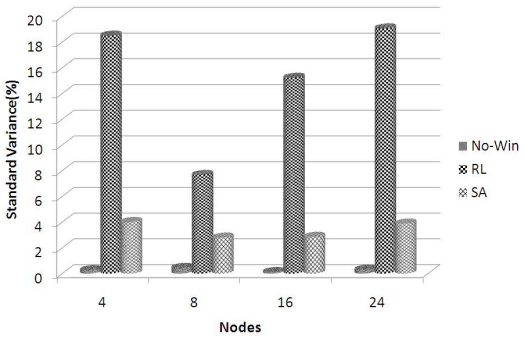

Figure 6. Standard Variance of Circuit C38417

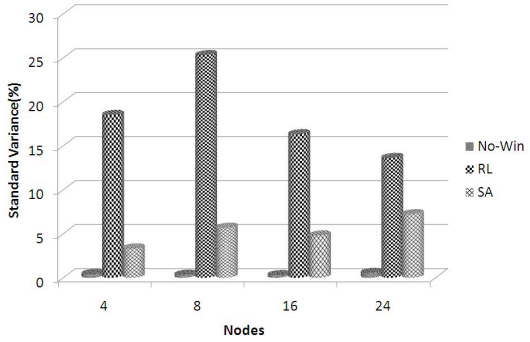

Figure 7. Standard Variance of Circuit C38417

RL algorithm is decided by the quality of candidates which are input to the algorithm, while the SA algorithm can find a good solution given a rather large choice of possibilities. As for the overhead, the overhead of both RL and SA is very low (no more than 1% of the computation time), so we do not portray detailed results for the overhead in this paper. The above experimental results clearly show that our SA algorithm can accelerate the simulation effectively as the speed-up obtained by our algorithm is better than any other existing method for choosing a time window.

V. Conclusion

In this paper, we presented a simulated annealing algorithm for choosing an optimal window size for bounded Time Warp. We utilized DVS[6] as our simulation engine and examined the performance of the SA algorithm for two circuits from the ISCAS benchmark suite. Our results showed that for different circuits and different topologies (i.e. differing numbers of processors), the SA algorithm can find a nearly optimal value for a time window. Our experimental results showed that the simulation time was reduced by up to 52% using this approach. To the best of our knowledge, this is the first time that SA has been utilized to chose a window for bounded Time Warp.

As for our future work, we plan to study the effect of adaptive simulated annealing for Time Warp simulation. Employing neural networks to optimize Time Warp is a future research direction as well.

References

[1] S. Palnitkar, *Verilog R hdl: a guide to digital design and synthesis*. Prentice Hall Press Upper Saddle River, NJ, USA, 2003.

[2] B. Cohen, *VHDL coding styles and methodologies*. Kluwer Academic Pub, 1999.

[3] F. Richard, "Parallel and Distributed Simulation Systems," *John Wiley & Sons, Inc*, vol. 605, pp. 10 158–0012, 2000.

[4] K. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Communications of the ACM*, vol. 24, no. 4, pp. 198–206, 1981.

[5] D. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985.

[6] L. Li, H. Huang, and C. Tropper, "Dvs: An object-oriented framework for distributed verilog simulation," in *Proceedings of the seventeenth workshop on Parallel and distributed simulation*. IEEE Computer Society Washington, DC, USA, 2003.

[7] H. Avril and C. Tropper, "Scalable clustered time warp and logic simulation," *VLSI design*, vol. 9, no. 03, pp. 291–313, 1998.

[8] Q. Xu and C. Tropper, "XTW, a parallel and distributed logic simulator," in *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society Washington, DC, USA, 2005, pp. 181–188.

[9] S. Meraji, W. Zhang, and C. Tropper, "On the Scalability of Parallel Verilog Simulation," in *The 38th International Conference on Parallel Processing, ICPP09, to appear*, 2009.

[10] B. Lubachevsky, A. Schwartz, and A. Weiss, "An analysis of rollback-based simulation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 1, no. 2, pp. 154–193, 1991.

[11] K. Panesar and R. Fujimoto, "Adaptive flow control in time warp," in *Parallel and Distributed Simulation, 1997. Proceedings. 11th Workshop on*, 1997, pp. 108–115.

[12] A. Palaniswamy and P. Wilsey, "Adaptive bounded time windows in an optimistically synchronizedsimulator," in *VLSI, 1993.'Design Automation of High Performance VLSI Systems', Proceedings., Third Great Lakes Symposium on*, 1993, pp. 114–118.

[13] J. Wang and C. Tropper, "Optimizing time warp simulation with reinforcement learning techniques," in *WSC '07: Proceedings of the 39th conference on Winter simulation*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 577–584.

[14] A. Barto and R. Sutton, *Reinforcement Learning: an introduction*. MIT Press Cambridge, MA, 1998.

[15] S. Kirkpartick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, Jan 1983.

[16] A. Kalai and S. Vempala, "Simulated annealing for convex optimization," *Mathematics of Operations Research*, vol. 31, no. 2, pp. 253–266, 2006.

[17] A. Xambre and P. Vilarinho, "A simulated annealing approach for manufacturing cell formation with multiple identical machines," *European Journal of Operational Research*, vol. 151, no. 2, pp. 434–446, 2003.

[18] Y. Nikulin, "Simulated annealing algorithm for the robust spanning tree problem," *Journal of Heuristics*, vol. 14, no. 4, pp. 391–402, 2008.