Logic Simulation Using T-Algorithm on Network of Workstations

S. Sundaram & M. K. Srinivas
Computer Aided Design Laboratory
Supercomputer Education & Research Centre
Indian Institute of Science
Bangalore 560 012.

Abstract

Increase in the complexity of VLSI digital circuit design demands faster logic simulation techniques than those currently available. One of the ways of speeding up existing logic simulation algorithms is by exploiting the inherent parallelism in the sequential version. In this paper, we propose a T-algorithm based logic simulation algorithm on a network of workstations interconnected by a local area network. The main objective has been to balance the computational load among the processors and at the same time reduce the communication to a bare minimum. We have achieved these by partitioning the circuit into cones containing gates in a fanout free region(FFR). Further, the cones of FFRs at the same level(having the same distance from primary inputs) are assigned by the master processor and communicated to the other workstations through the network. We have kept the balance on the computational load among slave processors by assigning a proper number of FFR cones, using a good heuristic. We have also shown that FFR partitioning reduces the amount of communication between the processors.

We have implemented the distributed logic simulation algorithm on a network of workstations using the communication primitives send and receive on sockets provided by the BSD 4.2 network library. The paper concludes with a tabulated results on the timings obtained on the ISCAS[8] benchmark circuits.

Keywords: Logic Simulation, Distributed Processing, Network of WorkStations, T-Algorithm.

Introduction

Logic simulation plays an important role as a verification tool in the process of VLSI design. Most widely used logic simulators employ the algorithm based on (1) Compiled code[1, 4] (2) Event driven[1, 4] or (3) T-algorithm[2, 4]. Logic simulation is extremely time-consuming; often for the large systems, we can only hope for partial simulation[11]. Even these partial simulations take enormous amount of CPU time, may be days or weeks. Exploiting the parallelism inherent in these algorithms and implementing them on parallel machines, helps in reducing the execution time of the simulation process[9].

Speedup can be achieved using Vector processors, Hardware accelerators [5] like YSE (IBM), LE (ZYCAD) or by mapping the simulation algorithm onto the processors of general purpose parallel machines or on a network of workstations.. In the case of vector processors, the code has to be rewritten to extract the computational power of vector processors(high vectorization ratio and long vector length)[3]. Hardware accelerators provide good speedups, but due to the direct mapping of simulation algorithm onto the hardware, it suffers from the disadvantage of cost to performance ratio, which turns out to be quite high. This makes the hardware accelerator, a less attractive scheme.

Current trends in carrying out parallel logic simulation are concentrated towards mapping the simulation algorithm onto a network of general purpose workstations connected on Ethernet. We exploit either functional parallelism which is inherent in the algorithm or data parallelism by dividing the circuit into subcircuits and assigning these to different processors[6]. The functional parallelism normally results in a pipeline of processors, each processor executing the subtask of the whole simulation task. In the latter case, load balancing and synchronization of simulation time are the vital issues which decide the partition of the circuit among processors[7]. The whole circuit is divided into cones of FFRs, and the cones in the same level are partitioned using a partitioning scheme, and each partition(set of cones) is assigned to a workstation for evaluation.

1 The Sequential T-Algorithm

In a T-algorithm(Time first evaluation algorithm), the evaluation of gates proceeds in the direction of signal flow i.e.from the input side towards the output side. The T-algorithm is based on the fact that the events associated with a gate can be evaluated independent of other gates for the whole simulation time period(in combinational circuits). During simulation, the evaluation of each gate advances asynchronously. The basic principle in the T-algorithm is to carry out the evaluation of a gate for the whole simulation period, for which the gate inputs are known, before the commencement of the evaluation of the next gate. Prior to the simulation execution, either the level sorting or the DF(Data Flow) sorting is used to find the order of gate evaluation. Level of a gate(cone) is the maximum distance in terms of the number of gates(cones) from the primary inputs. The primary inputs are at level zero. In the case of combinational circuits, the primary input gates are the first target gates for evaluation. The order of gates for evaluation, among the gates whose inputs are available can be random or can depend on some criteria like the number of fanins of the gate. Once these gates are evaluated, the outputs of these gates for the whole simulation period are known, and the fanout gates are selected for the next evaluation.

The process is carried out till all the gates in the circuit are evaluated.

The principal advantage on which the T-algorithm gains in simulation execution time over event driven simulation is that, once the gate is evaluated, the same gate is not referred again thereby saving the table lookup time. Because of this, the T-algorithm runs faster than the event driven algorithm for most of the combinational circuits[2]. In the case of synchronous sequential circuits and short feedback loop circuits the efficiency of the T-algorithm comes down depending on the interval length, but still performance can be improved using two path simulation technique[2].

2 Parallel Logic Simulation

The overhead of communication is a major bottleneck in parallel/distributed processing. If the ratio of computation/communication is kept high, we can execute algorithms faster in a parallel processing environment. The following paragraphs describes, how the T-Algorithm is tuned so as to get a maximum parallelism on a distributed environment. Since the unit of communication in the T-algorithm is a sequence of events unlike an event as in the event driven or compiled code simulation algorithms, the communication cost is kept low. Secondly, the computation is increased, as evaluation of the gate is for the whole simulation period and also the reference to a gate is made only once during the simulation process. Because of these reasons the T-algorithm based logic simulation performs well on a distributed environment. Secondly, it is desirable to increase the computation by evaluating *more number of gates per processor*, and reduce the communication time by communicating *only a few gate outputs* which are needed for the subsequent computation. Fan out free region partitioning allows to group the gates in the form of cone, through which we can evaluate a cluster of gates which are associated with every cone, and at the same time restricting the communication between worker task and master task to input wave forms and an output wave form along with the gate numbers associated with the cones. Further we have also carried out the load balancing among processors by dividing the cones among processors depending on the number of gates associated with the cones, such that all processors share equal number of gates for evaluation. This enabled us to increase the computation time and decrease the communication time. In our implementation, we have increased the computation/communication ratio further by evaluating a set of gates instead of a single gate and carrying out communication only once at the start and once at the end for the whole set of gates to be evaluated on a worker task[10]. The following subsections brief the partitioning and master worker abstraction of the simulation algorithm.

2.1 Partitioning of Gates among Worker Tasks

As mentioned in section 2 we have to maintain a high computation to communication ratio. To achieve this we initially partition the circuit into maximal fanout free regions(FFRs) in a preprocessing step. FFR partitioning is based on the principle that every maximal FFR output is either a primary output or a fanout stem and every FFR input is either a primary input or a fanout branch. The sequence of operations carried out in this algorithm is explained with the help of pseudocode in Figure 1. The FFR cones are levelized and all the cones in a particular level are evaluated in parallel. This technique effectively reduces the number of levels in the circuit as shown in Table 2, increases the number of gates and hence computation per processor, and decreases the communication among processors as only the output of the FFR cones have to be communicated to the master processor. Decrease in the number of levels also reduces the number of simulation cycles which in turn decreases the overall communication time. The partitioning strategy adopted in our present implementation is to assign FFR cones in the same level to different processors in such a way that almost equal number of gates are allotted to every processor to maintain a good load balance among the processors, as shown in Table 1.

2.2 Master and Worker Tasks

The worker and master task carry out reading of the circuit, forming the required data structure, evaluation of cones etc except that master task has to do an additional task of partitioning the circuit into cones using FFR partitioning technique, and divide the cones which belong to same level among master and worker tasks. A brief description of the routines and communication constructs present in the pseudo code of master task is as follows.

Rand_assign does the partioning of cones among workstations. **Form_buffer** sets up the array of data for communication, and these array of data are subsequently communicated to other workstations using receive and send constructs. After the communication of set of gates to the worker processors, the master task carries out the evaluation of gates which are assigned to it. After the evaluation, the master task receives the outputs from all workers, and updates the circuit data structure. This process is carried out in a loop till all levels (of FFR cones) in the circuit are exhausted. Referring to Table 1, the ten cones in level 2 are allocated to processors such that the three processors get 4, 2 and 4 cones with 5, 4 and 4 gates respectively. In most cases, the number of cones allocated in each level to a processor maintains the number of gates to be nearly equal except few cases like in level 9, there is only one FFR cone of 54 gates, that could not be partitioned further. Load balancing can be further improved if we consider the number of inputs associated with each gate and the length of the input list during the assignment to the processors.[12].

3 Results and Observations

Table 2 gives the characteristics of ISCAS85 benchmark circuits. Table 1 shows the partitioning of cones and gates among processors for a good load balancing. The FFR cone generation time is constant irrespective of the number of processors on which simulation is done and is only dependent on the number of gates in the circuit. The preprocessing time is also independent of the length of the simulation time and number of times the simulation is run. We can clearly observe from Figure 2 that the execution time reduces sharply for larger circuits(Ex: c1908) with the increasing number of processors. We also observed that our technique using FFR cones resulted in a good load balance and reduced communication among the workstations.

Table 1: Load Balancing for C432 using 3 Processors

Level #	*Cones*	*Processor 1*		*Processor 2*		*Processor 3*	
		Cones	*Gates*	*Cones*	*Gates*	*Cones*	*Gates*
1	8	3	10	3	12	2	8
2	10	4	5	2	4	4	4
3	1	1	1	0	0	0	0
4	3	1	1	1	1	1	1
5	9	3	3	3	3	3	3
6	9	3	9	3	8	3	9
7	1	1	1	0	0	0	0
8	3	1	1	1	1	1	1
9	1	1	54	0	0	0	0
10	2	1	1	1	58	0	0
11	7	2	12	3	12	2	8
12	3	1	2	1	2	1	11
13	3	1	3	1	3	1	1

4 Conclusions

In this paper we have described the suitability of the T-algorithm based logic simulation method on network of workstations. We have chosen to partition by fan out free regions, to decrease the number of levels and at the same time increase the computation by forming cluster of gates first and then associating cluster of cones to worker and master processors depending on the number of gates, to achieve a good load balancing (The processor running the master task is called master processor and the processors running the worker tasks are called worker processors). As we can see from the table the simulation time decreases with the increase in number of processors. The future direction will be to carry out the modification in the present implementation by refining the load balancing heuristic, and optimizing the code in the partition and simulation modules to give a still better performance.

References

[1] M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Rockville, 1976.

[2] N. Ishiura, H. Yasuura and S. Yajima, *Time First Evaluation Algorithm for High-Speed Logic Simulation*, Proc. ICCAD, November 1984, pp. 197-199.

[3] N. Ishiura, H. Yasuura and S. Yajima, *High-Speed Logic Simulation on Vector Processors*, IEEE Trans. on CAD, Vol. 6, No. 3, May 1987, pp. 305-321.

[4] S. Sundaram and L.M. Patnaik, *Logic Simulation Algorithms : An Overview*, Guest Editorial, Electronics & Telecommunications, Journal of Institution of Engineers, India, Vol 72, May 1991, pp. 1-7.

[5] Kitaomura. Y et.al, *Hardware Engines for Logic Simulation*, Logic Design and Simulation edited by E. Horbst, Elsevier Science Pub., NH. ,Advances in CAD for VLSI, Vol. 2, 1986.

[6] Srinivas Patil, Prithviraj Banerjee and Constantine D. Polychronopoulos, *Efficient Circuit Partitioning Algorithms for Parallel Logic simulation*, Supercomputing Conference 1989, pp. 361-370.

[7] Friedrich Hoppe, *Accelerated Logic Simulation using Parallel Processing*, Compeuro 1988, pp. 156-163.

[8] Franc Brglez, Philip Pownall and Robert Hum, *Accelerated ATPG and Fault Grading via Testability Analysis*, Proceedings of ISCAS 1985, pp. 695-698.

[9] S. Sundaram T.S. Mohan and L.M. Patnaik, *An Algorithm for Discrete Event Logic Simulation on Distributed Systems*, IEEE Region10 Conference 1991, VOL 3, pp. 185-189.

[10] S. Sundaram and L.M. Patnaik, *T-Algorithm-Based Logic Simulation on Distributed Systems*, Proc. of the Second Great Lake Symposium on VLSI, Michigan, Feb 28-29 1992.

[11] Larry Soule and Anup Gupta, *Parallel Distributed-Time Logic Simulation*, IEEE Design and Test of Computers, Dec 1989, pp 32-48.

[12] S.Sundaram and L.M. Patnaik, *Logic Simulation Algorithm on Network of Transputers*, Proc. of the 4th Transputer/Occam international conference, June 1992, Japan, pp. 219-230.

[13] Tom Blank, *A Survey of Hardware Accelerators used in Computer-Aided Design*, IEEE Design and Test of Computers, Vol 1, No. 3, pp. 21-39, 1984.

Table 2: Characteristics of the Combinational ISCAS Benchmark Circuits

Circuit	*Gates*	*Levels of gate*	*Levels of FFR cones*	*Cones/level*		*Gates/cone*	
				Min	*Max*	*Min*	*Max*
c432	160	17	13	1	10	1	54
c499	202	11	6	2	32	2	17
c880	383	24	13	2	21	1	23
c1355	546	24	16	1	39	1	17
c1908	880	47	19	1	79	1	37

```
{
 initialize stack;
 push primary output gates into stack;
 while (stack not empty)
   {
    do { /* for each gate in stack */
      pop the gate;
      if(visited gate) break;
      assign the gate the cone number;
      if(gate's fanin != 0)          {
          mark the gate as visited;
          for(every fanin of this gate)
            {
             if((not present in cone list) and (not visited))
               if((fanout < 2) or (fanin ==0))
                 push the gate to stack;
               else
                 store this gate in push_gate array;
            }
          include the gate in the cone list;
        }
      else /* it is a primary input */
        {
          mark the gate as visited;
          check whether the gate exists in cone list;
          if(! exist)
            attach the gate to cone list;
        }
    } while(current gate == next gate in stack);
    if(current gate == next gate in stack)
      {
       push current gate into stack;
       increment cone number;
      }
    push all push_num array gates into stack;
   }
 return(cone number);
} /* ffr algorithm ends */
```

Figure 1: Pseudocode for Fanout Free Region Algorithm

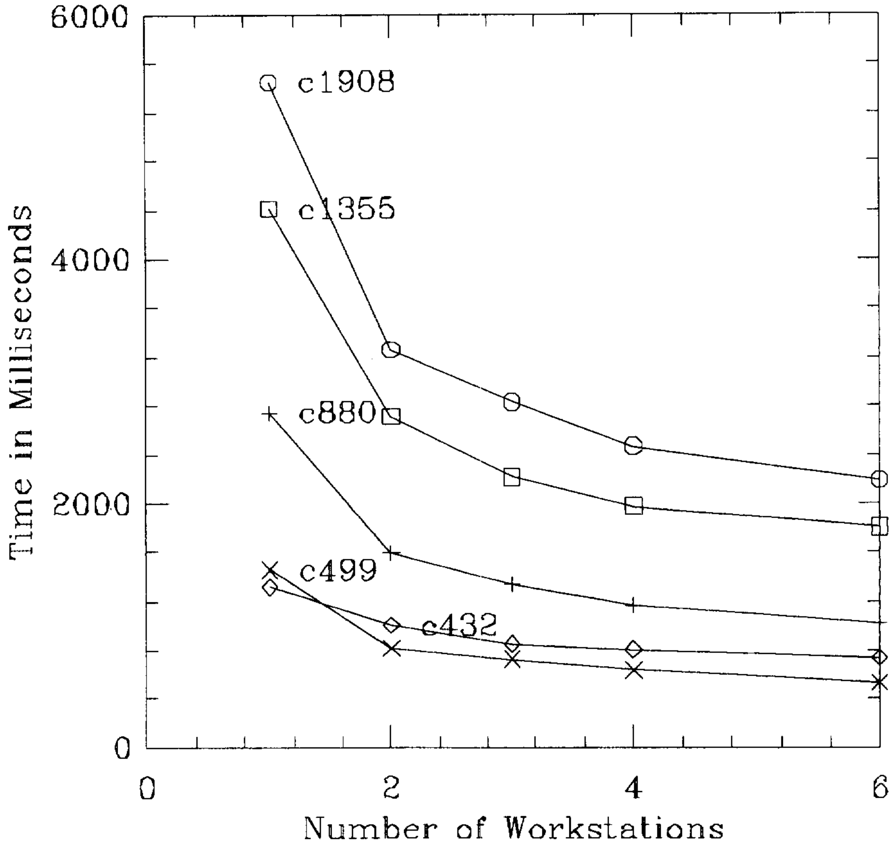

Figure 2: The Execution Time Curve obtained on ISCAS Benchmark Circuits