T-Algorithm-Based Logic Simulation on Distributed Systems

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

L.M. Patnaik
Microprocessor Applications Laboratory
Supercomputer Education & Research Centre
Dept. of Computer Science and Automation
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 explore the possibility of mapping a T-algorithm based logic simulation algorithm onto a cluster of workstations interconnected by an ethernet. The set of gates at a particular level is partitioned by the Master Task (running on the host processor) among the Slave Tasks(running on the other processors). Each Slave Task evaluates the set of gates assigned to it, for the complete simulation period independent of other slave tasks and communicates the evaluated outputs to the Master Task. After receiving the evaluated output from all the Slaves, the Master Task partitions the gates at the next level and communicates this new set of gates to the slave tasks. The above process is repeated for all the levels in the circuit. The details of the partitioning scheme and its performance are also discussed.

We have implemented this distributed logic simulation algorithm on a set of 8 VAX stations using the communication primitives 'send' and 'receive' provided as system calls. The paper concludes with a tabulated results on the speedup figures obtained on the ISCAS[8] benchmark circuits.

1 Introduction

Logic simulation plays an important role in the process of VLSI design. Existing sequential algorithms can be broadly classsified into three types namely (1) Compiled code[1,4] (2) Event driven[1,4] and (3) T-algorithm[2,4]. Logic simulation implemented using these algorithms takes considerable amount of time to carry out simulation on VLSI circuits. Exploiting the parallelism inherent in these algorithms and implementing them on a parallel processing environment, help in reducing the execution time of the simulation process[9].

One of the ways of obtaining speedup is by tuning the existing simulation code to run on vector processors. The code has to be rewritten to extract the computational power of vector processors(high vectorization ratio and long vector length)[3]. Using Hardware accelerators[5] like YSE(IBM), Le(ZYCAD) is another alternative method to achieve speedup in simulation process. In this case the parallel algorithm is mapped directly onto the hardware. This method gives good speedups, but suffers from the disadvantage that any change in the algorithm cannot be reflected back in the hardware easily and cost to performance ratio is high, thereby making it less attractive for speeding up logic simulation.

Current trends in carrying out parallel logic simulation are concentrated towards mapping the simulaton algorithm on the processors of general purpose parallel machines. 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]. Load balancing and synchronization of simulation time are the vital issues which decide the partition of the circuit among processors[7]. In our approach, we divide the logic gates at the same level into different partitions, and each partition is assigned to a processor for evaluation. The partitioning scheme can be varied to obtain better load balancing among processors. We have used the T-algorithm to run on a distributed enviornment, consisting of a cluster of 8 VAX 2000 workstations interconnected by Local Area Network as shown in Figure 1.

2 The Sequential T-Algorithm

Time first evaluation algorithm(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. In a T-algorithm, the evaluation of gates proceeds in the direction of signal flow i.e.from the input side towards the output side. 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. In the case of combinational circuits, the gates whose fanins are primary inputs, are the first target gates for evaluation. The choice of a gate for evaluation, among the gates which can be evaluated at the present instant can be random or can depend on some criteria like

0-8186-2610-0/92 $3.00 © 1992 IEEE

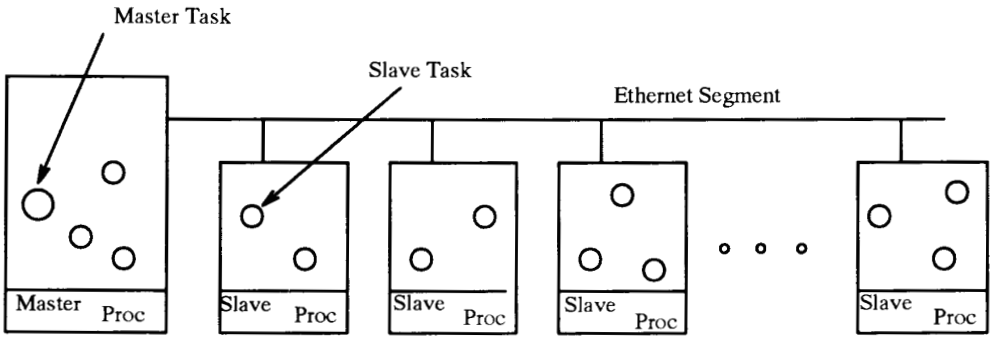

Figure 1: The Master-Slave Tasks on a Distributed System

the number of fanins of the gate. Once these gates are evaluated, the outputs of these gates for the whole simulation period are known. The gates which are connected to these outputs are selected for the next evaluation. The process is carried out till all the gates in the circuit are evaluated.

The principle 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 referenced again thereby saving the table lookup time. Because of this, T-algorithm runs 7 times faster than event driven for most of the combinational circuits[2]. In the case of synchronous sequential circuits and short feedback loop circuits the efficiency comes down depending on the interval length.

3 Parallel Logic Simulation

The overhead of communication is a major bottleneck in parallel processing. If the ratio of computation/communication is kept high, we can execute algorithms faster in parallel machines. Since the unit of communication in the T-algorithm is a sequence of events rather than an event as in other simulation algorithms, the communication cost is very low. Secondly, the computation is increased, as evaluation of the gate is for the whole simulation period and also the reference to the this gate is made only once during the simulation process. 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 slave. Though the evaluation time can be increased by having fast changing input pattern, this also gives raise to additional buffer setup and communication time. The performance of the algorithm for fast changing inputs needs to be studied in detail. The following section briefs the master slave abstraction of the simulation algorithm.

3.1 Master-Slave Tasking Abstraction

The evaluation of gates procedure from the simulation algorithm is seperated, as a slave task and this process run on the slave processor. The remaining parts of the code consisting of reading the input circuit description, finding the gates which can be evaluated, partitioning and communicating these gates to slave processors etc forms the master task. In the present implementation the master task does not take part in the evaluation of the gates. The sequence of operations carried out by the master task and slave task are described with the help of a pseudo code as shown in Figure 2 and Figure 3 respectively.

3.2 Partitioning of Gates among Slave Tasks

The partitioning startergy adopted in the present implementation is to assign gates in the same level to different processors in such way that almost equal number of gates are alloted to every slave processor. This scheme does not ensure good load balance among slave tasks, as it does not consider the number of fanins per gate. One of the better partioning schemes would be to consider the number of fanins of the gate so that the evaluation time will be uniform among processors. To arrive at the partition, number of fanins of the gate along with the input pattern length associated with each of these fanins have to be considered. Secondly, the communication can be minimized if the partitioning takes care of associating all the gates to a single slave processor, in case the number of gates at the current level fall short of some threshold value. Threshold value calculation involves, comparing computation vs communication costs by performing many simulation runs.

4 Results and Observations

The performance of the parallel T-algorithm depends mainly on the number of gates available at each level of the circuit for evaluation. Table 1 shows the minimum, maximum, average number of gates per level present in the ten ISCAS benchmark circuits. In addition, the table gives the average fanins[8] and number of levels, number of inputs and outputs in the circuits. Table 2 shows the timing of parallel logic simulation run for the ten ISCAS benchmark circuits

on a cluster of eight VAX station 2000 workstations interconnected by a LAN. The readings show the user and system time obtained by the system call 'times' during the simulation. Since the 'times' call returns variations in time value for different runs of the same circuit, each circuit was simulated ten times and the avearge readings are presented in Table 2.From the tabulated values, we can see that the efficiency of the algorithm increases with the increase in the number of slave processors, but after a certain value of slave processors, decrease in efficiency is observed.The time involved in communicating across the machines and synchronization are included in the system time. The exact amount of time spent on these operations has to be found out by some mechanism and added to the user time to get the correct value of time. Measurement of exact time incurred in communication, is a difficult parameter to calculate in a diskless enviornment due to network traffic present in the system.

Acknowledgements

The authors thank Prof.V.Rajaraman for his encouragement and the staff of KBCS, CADL, MAL and SERC for their help. The authors like to convey special thanks to their colleagues M.K.Srinivas and D.Sampath for clarifying some important points through discussions, during the implementation.

References

[1] M.A. Breuer and A.D. Friedman, *Diagonsis 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.

```
master_task() /* master task begin */
{
#define NUM_PROC 3  /* Number of processors */
int sd[NUM_PROC];  /* Socket for communication */
int maxdist; /* depth of the circuit */
   /* Global Data Structure */
main()
{
  int i,j;  /* local data structure */
   /* Read the circuit description and input stimulus */
   host_accept  /* Establish connection with slaves */
   /* Evaulation time counter begins */
   for(i=1;i<=maxdist;i++)
     {
       rand_assign();  /* Assignment of gates among slaves */
       form_buffer();  /* Creation of buffer to transfer data to
                          slaves */
         /* Communicating to slaves */
      for(j=0;j<NUM_PROC;j++)
         {
            send_int_arr(j,buffer,num_int); /* header information */
            send_int_arr(j,buffer,number); /* Actual circuit details */
         }
       /* Receive the outputs from slaves which were active */
          /* Depends on the partition */
      for(j=0;j<work_proc;j++)
          {
             recv_int(j,num_int);  /* header information */
             recv_int_arr(j,buffer,num_int); /* Gate evaluation outputs */
             recv_update(); /* Update the local data structure */
          }
      } /* Simulation end */
     /* Evaulation time counter ends */
     /* Termination broadcast to slaves */
     /* Printing the required results */
}
}/* master task end  */
```

Figure 2: Pseudo code for Master Task

Table 1: Characteristics of the Combinational ISCAS Benchmark Circuits

Circuit	*Levels*	*Gates/level*			*Gates*	*Fanins*		*Inputs*	*Outputs*
		Max	*Min*	*Avg*		*Avg*	*Max*		
c432	17	27	1	8	160	2.10	9	36	7
c499	11	40	2	16	202	2.02	5	41	32
c880	24	53	3	15	383	1.90	4	60	26
c1355	24	64	2	21	546	1.95	5	41	32
c1908	47	88	1	18	880	1.70	8	33	25
c2670	42	80	1	27	1193	1.74	5	233	140
c3540	60	168	1	27	1669	1.76	8	50	22
c5315	60	171	1	37	2307	1.90	9	178	123
c6288	124	256	1	19	2406	1.99	2	32	32
c7552	54	275	1	63	3512	1.75	5	207	108

```
slave_task() /* slave task begin */
{
  int truthtab [6][5][5] /* For the truth table */
  /* Global data structure */
  int s1; /* socket at the slave side */
main()
 {
   int i,l; /* local data structure here */
   slave_connect /* Establishing connection with master */
   while(1) /* Infinite loop */
    {
       recv_int_arr(s1,buff,num); /* receive header from master */
       if(Term_signal) break; /* Termination detection */
       recv_int_arr(s1,buff,number); /* Receive circuit details from
                                        master */
       for(l=0;l<num_gates;l++) /* Evaluation of set of gates */
          evaluate_gate(l);
       /* communicating back the results */
       send_int(s1,number); /*header info for the master */
       send_int_arr(s1,buffer1,number); /* evaluated outputs */
     }
   close(s1);
 }  /* slave task end */
```

Figure 3: Pseudo code for Slave Task

Table 2: Parallel Logic Simulation Timings

Circuit	*Number of Processors*											
	1		*2*		*3*		*4*		*6*		*8*	
	Time(ms)		*Time(ms)*		*Time(ms)*		*Time(ms)*		*Time(ms)*		*Time(ms)*	
	usr	*sys*	*usr*	*sys*	*usr*	*sys*	*usr*	*sys*	*usr*	*sys*	*usr*	*sys*
c432	1025	577	615	1903	615	2260	607	2292	627	2197	642	2565
c499	1032	543	575	1968	617	2163	635	2182	627	2357	606	2513
c880	2115	1072	1308	4607	1325	5193	1395	5178	1390	5528	1443	5677
c1355	3477	1338	2027	6992	2056	7550	2033	7702	2058	7952	2088	8263
c1908	4573	2321	2948	9552	2980	10785	3025	10917	3060	11553	3183	11947
c2670	6035	2792	3961	13130	4035	14087	4083	14125	3993	14235	4159	14912
c3540	10091	3941	6228	20067	6252	21505	6247	21940	6365	22562	6433	23225
c5315	15550	4917	9010	28823	8981	30405	9060	30735	9137	31517	9237	32007
c6288	21963	6538	12895	41642	12948	42307	13342	42537	13425	43828	13512	45095
c7552	24393	7185			14108	50710	14423	51227	14411	52040	14462	52852