ICOT Technical Memorandom: TM-1030

TM-1030

Parallel Logic-level Simulation System on a Distributed Memory Machine

by
Y. Matsumoto & K. Taki

February, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Parallel Logic-level Simulation System on a Distributed Memory Machine

Yukinori MATSUMOTO Kazuo TAKI
Institute for New Generation Computer Technology

January 11, 1991

Abstract

We have just constructed a parallel logic-level simulation system based on Virtual Time. It was implemented on the Multi-PSI, the experimental parallel inference machine developed at ICOT. Our system simulates sequential circuits of practical size. Different delay times can be assigned to gates in the circuits.

This paper proposes a new partitioning strategy of circuit data that is efficient for Virtual Time and also reports the measurement results of the simulator. In our experiment, a sequential circuit consisting of over 10,000 gates was simulated. Using 64 processors, the system indicated about 47k events/sec as its performance, and also indicated about 47 times speedup. This paper, using these results, denotes that Virtual Time is an efficient local synchronization mechanism and the new partitioning strategy proposed here can be a practical strategy.

1 Introduction

Logic-level simulation is one of the most important stages in LSI design. It verifies both logic design of circuits and timing of signal propagation in the circuits. Since the simulation phase consumes a lot of computation time, faster simulators are urgently needed. Although a hardware simulator works quickly, it has little flexibility in actual use. A parallel software simulator is a likely alternative. Many people expect it to be useful when accurate simulation is required.

We have constructed a parallel logic-level simulation system based on Virtual Time on the Multi-PSI, an experimental parallel inference machine. This system is one of the target applications of the Fifth Generation Computer Project.

This paper firstly overviews our system. A local message scheduler and antimessage reduction mechanism are mentioned. Secondly, this paper argues a load-balancing strategy useful for a simulator based on Virtual Time. A new partitioning strategy named "the Depth-first Search Oriented Partitioning Strategy" is proposed. Finally we report the performance and speedup of our system in actual execution.

2 Event Simulation and the Virtual Time Paradigm

The simulation mechanism of our system is classified as event simulation. Event simulation can be represented by a model in which several objects change their states by communicating with each other. An object is described as a state-automaton. A message has information of an event whose occurrence time is stamped on the message (time-stamp). An object changes its state after receiving a message from another object. If a new event arises, the object also sends messages to other objects that the event concerns.

In this model, messages should arrive at their destination objects in time-stamp order to maintain correctness of the simulation. So usually a centralized message scheduling mechanism is required for sequential execution. This mechanism is called the time wheel. In parallel execution, however, a local synchronization is more suitable because such a centralized mechanism decreases parallelism considerably.

Chandy and Misra proposed a local synchronization mechanism based on the block-and-resume mechanism [2]. Though fairly efficient, its chief defect is that its naive implementation always runs the risk of deadlock. Generally it takes a lot of overhead to avoid deadlock.

Jefferson proposed the Virtual Time Paradigm and its implementation, the Time Warp Mechanism [1]. As it uses no blocking mechanism, there is no possibility of deadlock.

In Virtual Time, assuming that messages might arrive chronologically, each object is usually executing and also recording the history of messages and states. But when a message arriving at an object in incorrect order, the object rewinds its history (this procedure is called rollback). Then it executes again from the time when the message should have arrived. If there are several messages which should not have been sent, the object also sends antimessages in order to undo, or cancel, those messages.

As frequent rollback occurrence decreases the total performance of the system, it is necessary to reduce the incidence of rollback.

3 System Overview

This system is written in a parallel logic programming language KL1 on the Multi-PSI. Multi-PSI is a MIMD machine where 64 processing elements are connected to each other to form a 2-dimensional mesh.

The Multi-PSI is a distributed memory machine, so it costs rather a lot to access data in other processing elements, but it is also easy to scale up.

KL1 is a stream AND-parallel logic programming language. In stream AND-parallelism, processes are executed concurrently, communicating with each other through the message streams. There is a lot of pipeline parallelism.

This system simulates combinatorial circuits and sequential circuits that have loop structures and states in the loop. It handles three values: Hi, Lo, and X. Multiples of a unit time can be assigned to each gate as its delay time. This system is very simple, having only essential functions for the experiments of parallel processing, but we can easily add other functions if necessary.

3.1 Local Message Scheduler

In our system, a gate corresponds to an object and a signal line corresponds to a channel. Since there are usually many more objects than processors, each processor has to take charge of several objects. For this reason, scheduling messages locally in each processor is expected to reduce rollback frequency efficiently.

In our system, each processor has a message scheduler for that purpose. When a message is spawned, the message is not sent to its destination object directly. Instead, the message is first sent to the scheduler which manages its destination object. The scheduler sends the message to its destination object at a suitable time.

Basically a scheduler behaves in a way similar to the time wheel of sequential event simulation. The scheduler has its clock value like the time wheel. The clock value is defined as the smallest time-stamp value of messages that are registered at that moment.

The difference between the time wheel and the message scheduler is that the clock value of a time wheel increases monotonically, whereas the clock value of a message scheduler sometimes decreases. Decreasing occurs when a message with smaller time-stamp value than the scheduler clock arrives from another processor.

3.2 Reduction of Antimessages

If we follow the original Virtual Time, when rollback occurs, an antimessage must be generated for each message that needs to be undone.

However, the number of antimessages can be reduced when we assume the following two conditions.

Environment conditions

1. The network of objects is static.

2. Message sending order is kept when messages are received by their destination objects.

Assume $M_1, M_2, ..M_n$ are messages and AM is an antimessage. Also assume that messages $M_1, M_2, ..M_n$ satisfy these three conditions just before the antimessage AM is sent.

Undone message conditions

1. They were sent before the antimessage AM.

2. They were sent along the same channel that the antimessage AM will be sent along.

3. They have time-stamp values greater than or equal to the antimessage AM.

It is clear that $M_1, M_2, ..M_n$ should be undone; no other messages should be undone.

Only the antimessage that corresponds to the undone message with the smallest time-stamp value need be sent. The destination object can recognize that all the messages satisfying the undone message conditions should be undone.

In our system, target circuits have static networks. However, the second environment condition may be not satisfied because messages are sorted by the message scheduler. So the messages are stamped in the order in which they were sent (order stamp), in addition to being time-stamped. Since the second environment condition is satisfied by using the order-stamp, our system needs fewer antimessages.

4 Partitioning Strategy

In parallel logic-level simulation, since the amount of work per message at an object is very small (the grain size of processing is very small), inter-processor communication cannot be ignored. Besides, as there are speculative computations during simulation based on Virtual Time, rollback happens sometimes.

When partitioning of a given circuit network is done, we have to consider the following three points so that simulation will be executed efficiently on a distributed memory machine.

1. Load balancing should be nearly equal
2. Inter-processor communication frequency should be reduced
3. The incidence of rollback should be reduced

Theoretically we should define an evaluation function that expresses the above three requirements adequately. Then we have to find a partitioning solution that attains the best function value. Since such a problem is considered NP hard, some heuristic algorithms are required.

In this paper, we propose a new partitioning strategy named "the Depth-first Search Oriented Partitioning Strategy" (in short DSOPS). This strategy is expected to satisfy the above three points tolerably. It is also expected to require only a little computation time.

DSOPS intends to demarcate clusters in which gates are connected in a cascade form. The algorithm is outlined below.

1. Select gates connected to input terminals of the circuit. Those gates will be the start points of search
2. Form the gates into a queue
3. Choose a new gate that does not belong to any clusters. It is called the current gate and will be used as a start point. If there is no such gate in the queue, then finish.
4. Allocate some memory area ready to hold a new cluster
5. Assign the current gate to that memory area
6. List all gates that connect the output of the current gate, except the gates already included in other clusters. If there is no gate listed, go to 3.
7. Pick an arbitrary gate from the gates listed in 6, and reguard that gate as the current gate.
8. Enqueue the rest of the gates as new start points of search, and go to 5.

After the procedure is finished, there may be small clusters that contain very few gates. They should be merged into large clusters that the small ones are connected to. Conversely, extremely long cascade-formed clusters should be cut into several small clusters.

Finally clusters are assigned to processors randomly; the only constraint is that each processor should contain a roughly equal number of gates.

DSOPS is expected to be a practical strategy because it give us a fairly good solution with only a little computation time.

Number of PEs	2	4	8	16	32	64
Inter-processor channels ratio(%)	10.02	14.02	15.99	16.90	17.22	17.43

Table 1: Ratio of inter-processor channels

Number of PEs	Performance (events/sec)	Speedup (times)	Ratio of rollback messages (%)	True message balance (%)	Upper limit of speedup
1	986	1.0		—	—
2	1,852	1.878	1.33	102.33	1.95
4	3,545	3.595	4.05	103.27	3.87
8	7,090	7.191	7.01	106.10	7.54
16	14,331	14.534	7.89	116.37	13.75
32	28,209	28.619	11.33	118.83	26.93
64	46,666	47.329	24.65	139.09	46.01

Table 2: Measurement results of simulation

5 Result and Discussion

We executed several experimental simulations on the Multi-PSI. We used data of a sequential circuit "s13207.bench", which was published by ISCAS'89. We partitioned it according to DSOPS.

In our experiments, the clock cycle was fixed to be a constant value. Signals to the other input terminals were given randomly. They changed synchronously with the clock rising time.

Table 1 shows the percentage of all channels that are being used as inter-processor channels. That percentage represents quality of the partitioning result in terms of reducing inter-processor communication. PEs means processing elements.

Table 2 shows several clues that suggest the performance of our system. Ratio of rolled back messages means the ratio of all the rolled back messages to all the true messages. True message balancing suggests degree of load imbalance on the assumption that all the computation times of true messages at gates are equal. The value of true message balancing is calculated by $\text{Num\_of\_Msg}_{max}/\text{Num\_of\_Msg}_{avr}$; where Num_of_Msg_{max} is the number of true messages received by the scheduler that received most true messages, and Num_of_Msg_{avr} is the average number of true messages received by schedulers. Upper limit of speedup is calculated by (number of PEs)/(true message balancing). It denotes the upper limit of speedup when the influence of load imbalance is taken into account.

Figure 1 shows the speedup of our experiments. The vertical axis shows speedup and the horizontal axis shows the number of processors.

Table 1 claims that DSOPS is efficient for reducing the number of inter-processor channels. The number of *intra*-processor channels is expected to be greater than the number of gates in the circuit when partitioned according to DSOPS. Since there are 18,489 channels of s13207.bench and 11,965 gates, the ratio of inter-processor channels to all channels should

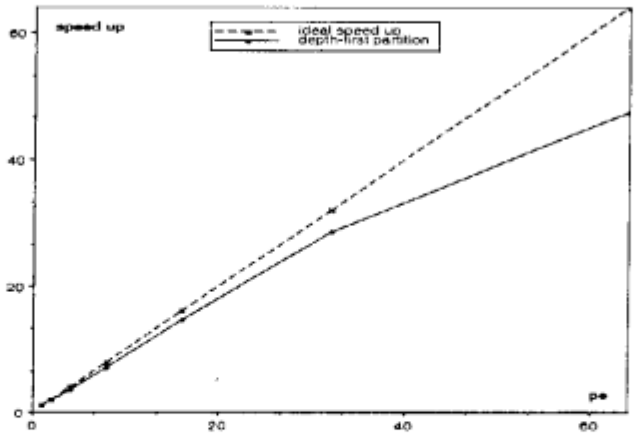

Figure 1: Speedup

be 35.29% at most. In practice, the ratio is 17.43% in case of 64 partitioning, much less than the maximum value 35.29%. The merging procedure, in which small clusters are merged into large clusters, might reduce inter-processor channels.

Figure 1 shows that linear speedup was attained using up to 32 processors, whereas the speedup using 64 processors was a bit worse then expected. Generally these three reasons affect the speedup.

1. High cost of inter-processor communication
2. Frequent occurrence of rollback
3. Load imbalance

Table 2 shows that actual speedup is roughly equal to the speedup limit. It means that load imbalance should have had the dominant effect on speedup in this experiment. In the processor where extremely many events occur, simulation speed is the slowest among all processors. Rollback rarely occurs in the processor, but it decides the total speed of simulation.

Generally speaking, the larger the number of clusters (and therefore the smaller the cluster size), the better load balance, when clusters are distributed to processors at random. Since the load imbalance appears to be the major performance limitation in our experiment, we can expect to improve performance by slightly increasing the number of clusters. Too many clusters, however, would cause too much communication overhead.

Now let us consider the influence of rollback. As we discussed above, rollback does not affect the total speedup when we use 16 or more processors. Rollback occurs only in processors where simulation proceeds too fast. When we use eight or fewer processors, although both inter-processor communication and rollback affect speedup, their influence is not so serious.

Some say that Chandy's algorithm suffers from having an enormous number of null messages, and Virtual Time suffers from rollback. Our results, however, show that rollback does not happen so often, and when it does happen, it does not affect speedup seriously. Virtual Time has potential as a local synchronization mechanism for parallel logic-level simulation.

6 Conclusion

We constructed a parallel logic-level simulator. We also measured the performance of our system using circuit data partitioned based on the Depth-first Search Oriented Partitioning Strategy. By evaluating the performance of our system, we conclude that DSOPS gives comparatively good partitioning solution. We also recognized that rollback did not decrease the total performance of the simulator and that load imbalancing was the main factor affecting speedup. Nearly linear speedup was attained in our experiment.

In the future, we will compare DSOPS with other strategies. We also intend to analyze the relation between frequency of rollback and partitioning results using various benchmark data.

References

[1] D.R.Jefferson, "Virtual Time," ACM Transaction on Programming Languages and Systems,Vol.7,No.3, 404-425 (1985)

[2] J.Misra, "Distributed Discrete-Event Simulation," Computing Surveys,Vol.18,No.1, 39-64 (1986)