ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Simulation Modelling Practice and Theory xxx (2005) xxx–xxx

SIMULATION MODELLING PRACTICE AND THEORY

www.elsevier.com/locate/simpat

Towards large scale optimistic VLSI simulation

Qing Xu *, Carl Tropper

School of Computer Science, McGill University, 3480 University Street, McConnell Engineering Building, Montreal, Que., Canada H3A 2A7

Received 16 September 2005; accepted 14 October 2005

Abstract

In this paper, an optimistic parallel and distributed logic simulator, XTW, is proposed. In XTW, a new event scheduling mechanism, XEQ, and a new rollback procedure, rb-messages, are proposed for use in optimistic logic simulation. XTW groups LPs into clusters, and makes use of a multi-level queue, XEQ, to schedule events in the cluster. XEQ has an O(1) event scheduling time complexity. Our new rollback mechanism replaces the use of anti-messages by an rb-message, and eliminates the need for an output queue at each LP. Experimental comparisons to Clustered Time Warp reveal a superior performance on the part of XTW, while experimental results on large circuits (5-million-gate to 25-million-gate) demonstrate that XTW scales well with both the size of a circuit and the number of processors used in the simulation.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Time Warp; VLSI simulation; Parallel and distributed simulation; Logic simulation; Event queue; Discrete event simulation

1. Introduction

In this paper, a new optimistic synchronization mechanism, XTW [1], is proposed in an effort to improve the performance of Time Warp [2] in the domain of VLSI simulation. In Time Warp causality errors are corrected by rolling back the state of the simulation to a previous correct state and eliminating erroneously sent messages and their effects by the sending of anti-messages. XTW was inspired by several characteristics of discrete event logic simulation. XTW has a new event scheduling algorithm, XEQ, and a new rollback mechanism, rb-messages.

Discrete event simulations of circuits at the logic, behavioral and register-transfer level all exhibit the following characteristics:

(1) Events generated by an LP are produced in chronological order (see Fig. 1).
(2) In a parallel discrete event simulation which uses a communication facility guaranteeing FIFO order, messages from a source LP arrive at a destination LP in chronological order (Fig. 1).

* Corresponding author. Tel.: +1 514 3987071x0834; fax: +1 514 2213696.
E-mail addresses: qing.xu@mail.mcgill.ca, qxu2@cs.mcgill.ca (Q. Xu), carl@cs.mcgill.ca (C. Tropper).

1569-190X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2005.10.003

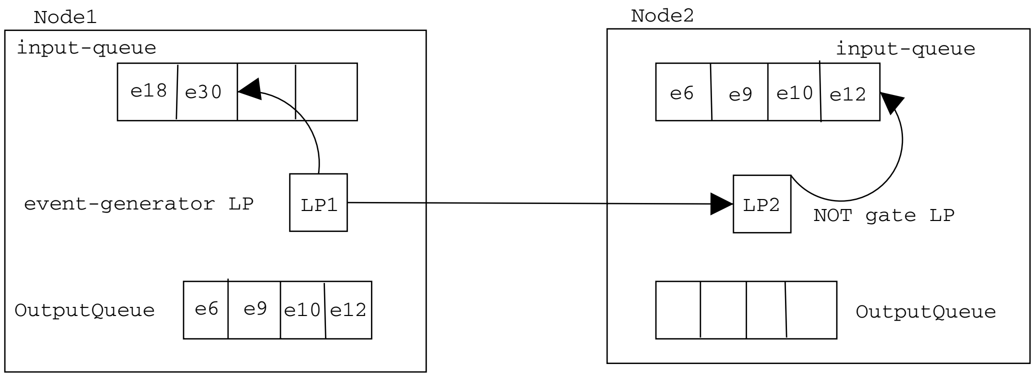

Fig. 1. A single LP, single input channel model in PDES.

(3) LPs are sparsely connected.
(4) The LP topology is static during the simulation.

Observations 1 and 2 show that events are naturally sorted when they are generated and propagated. These observations are the keys to our approach and inspire us to create a new multi-level queue, XEQ, accompanied by an event scheduling algorithm which utilizes these naturally sorted events. We make use of XEQ to create the rb-messages mechanism in order to reduce the rollback cost in Time Warp. Observations 3 and 4 make it feasible to implement XEQ and rb-messages in a large circuit simulations.

We explore how to utilize the naturally sorted (i.e. zero-cost) events in discrete event circuit simulation. First, we examine a parallel logic simulator using Time Warp. One LP is an event-generator and resides on Node 1, while another component is a NOT gate residing on Node 2. The LPs communicate with each other via a FIFO communication facility (Fig. 1). In Fig. 1, we can see that the event scheduling cost at Node 1 is O(1), consisting of the cost to append the generated events to the input queue and to de-queue the head event from the input queue. At Node 2, the event scheduling cost is also O(1), consisting of the cost to append the events coming from Node 1 to the input queue and to de-queue the head event from the input queue.

When there are a large number of LPs residing on one node and multiple input sources on one LP, the situation is totally different. Events generated by different LPs and coming from different sources have to compete with each other in order to be inserted into the input queue (Fig. 2). In Time Warp, an LP can be rolled back and generate out of order events, further complicating matters. In the next section we describe a data structure, XEQ, which makes it is possible to preserve the zero-cost sorted events and has an O(1) event-scheduling cost. A new rollback mechanism, rb-messages, is proposed to reduce the rollback overhead.

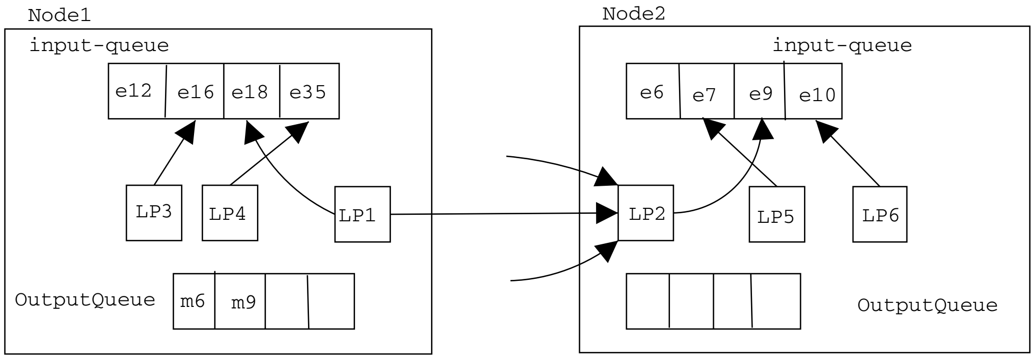

Fig. 2. A multiple LPs, multiple input channels model in PDES.

The remaining sections of this paper are organized as follows. Section 2 contains a detailed description of XEQ and rb-messages along with an analysis of their complexity. Section 3 contains our experimental results. Section 4 describes related work. The concluding section of the paper follows. Appendix A contains brief descriptions of four optimization techniques which are made use of in XTW.

2. XTW

XTW makes use of clusters of LPs. The clusters represent groups of gates which belong to the same functional unit. Each cluster has a multi-level event queue, XEQ, associated with it. A cluster level event queue (CLEQ) which is part of XEQ stores events which are sent to other clusters.

XTW is an outgrowth of Clustered Time Warp [3]. Three techniques for checkpointing and rolling back in Clustered Time Warp are described in [3], each occupying a different point in a memory vs. execution time trade-off. XTW makes use of one of these techniques, local rollback,local checkpoint. Local checkpoint means that an LP saves its state only if it receives a message from an LP in another cluster. Local rollback refers to each LP rolling back individually (as opposed to requiring that all of the LPs in a cluster roll back- another technique in Clustered Time Warp).

This section contains a detailed description of XEQ and the rb-message mechanism, along with an analysis of their complexity. We organize the section as follows. Section 2.1 introduces the input channel structure. Section 2.2 presents the structure of XEQ. Section 2.3 contains the event node structure. Section 2.4 presents the XTW event scheduling mechanism and its cost analysis. Section 2.5 presents the rb-messages mechanism.

2.1. Input channel

In XTW, a new structure, the input channel is added to LPs. Each input channel models an unique input of a circuit component and is subject to *Rule 1* as follows:

- *Rule 1: Each input channel can only have one unique incoming source.*

Fig. 3 shows how the input channel models the connection edge of gates.

Fig. 4 shows the structure of the input channel. Each input channel contains one input event queue (ICEQ) and one processed event queue (ICPQ). Newly arrived events are put in the ICEQ. After an event is processed, it is put in the ICPQ. Each event has the following time-stamps: (a) a receive-time stamp which indicates when the event occurs (b) the send-time stamp which is the Local Virtual Time of the LP when it scheduled the event. The LVT is the receive-time of the latest processed event.

As a result of observations 1 and 2 and *Rule 1*, all of the normal events must arrive at each ICEQ in chronological order and be naturally sorted in the ICEQs (Fig. 4).

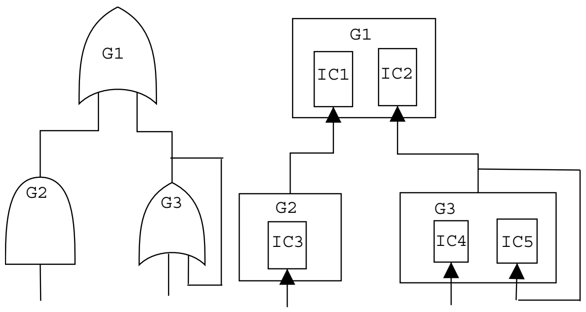

Fig. 3. Input channel model.

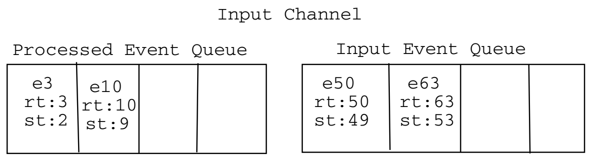

Fig. 4. The structure of input channel.

2.2. The structure of XEQ

Fig. 5 shows the structure of XEQ. In XEQ, there are event queues at the input channel level, the LP level and the Cluster level.

- At the input channel level, the event queue is called the *ICEQ* and is implemented as a list of events sorted in increasing time-stamp order.
- At the LP level, the event queue is called the *LPEQ* and is implemented as a list of events sorted in increasing time-stamp order.
- At the cluster level, the event queue is called the *CLEQ* and is implemented as a list of time-buckets sorted in increasing time-stamp order. A time-bucket is a list of events which have the same time-stamp.

In addition, the following event pointers are added respectively for each input channel and each LP.

- CIE: At each input channel, a CIE (current-IC-event) pointer points to the event which is de-queued from its ICEQ and is currently stored in the LPEQ or the CLEQ. This pointer is used to remove the (pointed-to) event from the LPEQ or the CLEQ in the event of rollback.
- CLE: At each LP, a CLE (current-LP-event) pointer points to the event which is de-queued from its LPEQ and is currently stored in the CLEQ. This pointer is used to move the (pointed-to) event from the CLEQ back to LPEQ in the event of a rollback at the LP.

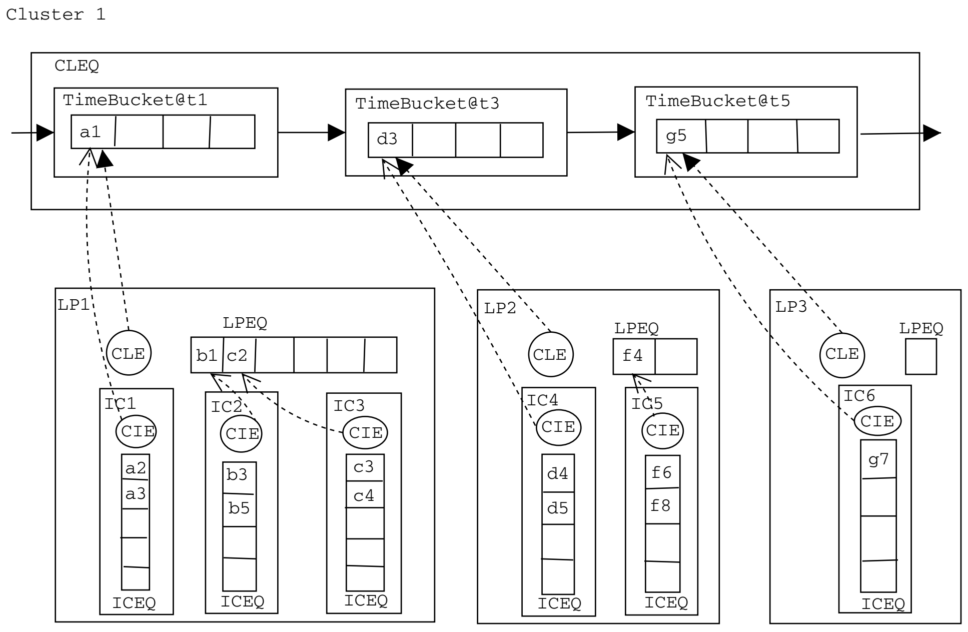

Fig. 5. The structure of XEQ.

2.2.1. Rules for XEQ

The following rules are enforced in XEQ:

- *Rule 2: An input channel can submit only one event to its hosting LP's LPEQ if and only if the ICEQ is not empty. This event has the lowest time-stamp in the ICEQ and is called the current IC event. Its pointer value is assigned to CIE.*
- *Rule 3: An LP can submit only one event to its hosting cluster's CLEQ if and only if the LPEQ is not empty. This event has the lowest time-stamp in the LPEQ, It is called the current LP event and its pointer value is assigned to CLE.*

2.3. Event node structure

Fig. 6 shows the structure of an event node and how an event node moves around among the different levels of the event queue.

Moving an event node from one event queue to another event queue is accomplished by changing the values of the next and the prev pointer of the event node. No copying is necessary and as a consequence, extra memory is not required at each of the event queues. An example is depicted in Fig. 6. When e1 is moved from the ICEQ to the LPEQ, the next and prev pointers of e1 are changed from I1,I2 to L1, L2. Similarly, moving e1 to the CLEQ or ICPQ involves changing the next and prev pointers to C1, C2 or P1, P2.

XEQ can be viewed as a Time Warp input queue broken into small pieces. The total space cost of XEQ is approximately the same as that of the Time Warp input queue structure.

2.4. XTW O(1) event scheduling mechanism

XEQ is used to implement a smallest time-stamp first event scheduling mechanism within clusters which has an O(1) time complexity.

An event is scheduled and processed in XTW via the following steps:

(1) After an event is generated, it is propagated to its destination input channel and is appended to the ICEQ.

(2) According to *Rule 2*, if the ICEQ is not empty it will place the smallest receive-time event to its LPEQ. Since the ICEQ is naturally sorted, the smallest time-stamp event is the head event of the ICEQ. Thus, we can de-queue the head event at a cost of 1.

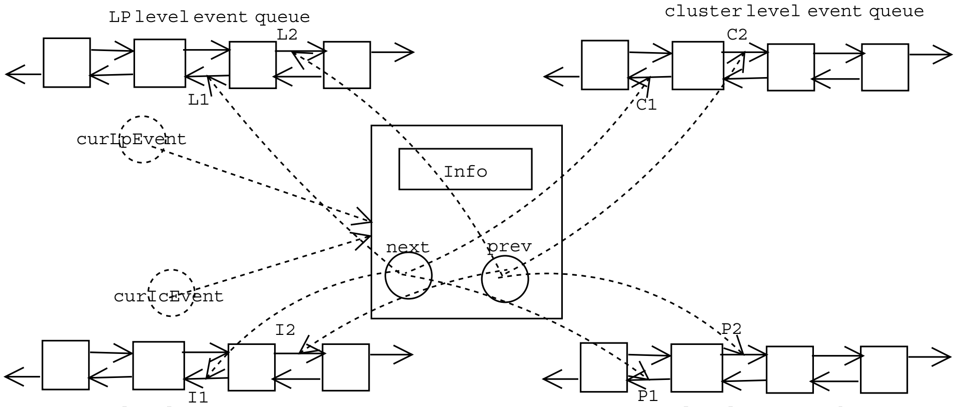

Fig. 6. An event node structure and its movement.

(3) The event from the ICEQ is inserted into the LPEQ. The cost of finding the correct position into which to insert the event is N_e. N_e is the number of events stored in the LPEQ. From *Rule 2*, in the worst case the maximum value of N_e is C_{ic}, where C_{ic} is the number of input channels at an LP.
(4) According to *Rule 3*, if the LPEQ is not empty, it will submit the head event to the CLEQ. The cost of finding the correct position in the CLEQ is N_{tb}, where N_{tb} is the number of time-buckets in the CLEQ. From *Rule 3*, in the worst case the maximum value of N_{tb} is C_{lp}, where C_{lp} is the constant number of LPs in a cluster.

Putting the above observations together, the cost of scheduling an event in XTW, *SC*, is

$$SC = 1 + N_e + N_{tb} \tag{1}$$

In the worst case the cost of scheduling an event is

$$SC = 1 + C_{ic} + C_{lp} \tag{2}$$

Since both C_{ic} and C_{lp} are constant, the complexity of scheduling an event is O(1). In reality, C_{ic} is far less than C_{lp} in most discrete event circuit models and making use of an $O(\log N)$ data structure in the CLEQ, results in an event scheduling cost of $O(\log C_{lp})$.

Comparing XEQ to other event-list data structures we first note that their time complexity is bounded by the number of events in the queue. Standard event list structures and their time complexities include the calendar queue (O(1)), the splay-tree ($O(\log n)$), the red-black tree ($O(\log n)$), the skip-list ($O(\log n)$) and the heap ($O(\log n)$). XEQ has more stable performance because it is bounded by the number of LPs, which is static during the simulation. It is not sensitive to the distribution of events as is the calendar queue. Moreover, XEQ can be used in both parallel and sequential discrete event circuit simulation and is easily implemented.

2.5. Rollback with rb-messages

We begin with the 2 LP example shown in Fig. 1. We assume that a rollback occurs at LP1- event e8 is generated after e12 in LP1 and is sent to LP2. In Time Warp anti-messages for e9, e10 and e12 are sent out to annihilate the events in LP2. However, in this example LP2, upon the arrival of e8 can annihilate e9, e10 and e12 without the necessity of anti-messages. Consequently, the output queue can be eliminated from each LP, since no anti-messages are required to annihilate the previously sent messages.

The advantage of above scenario is obvious—we cannot only reduce rollback overhead by eliminating anti-messages, but can also save memory by not saving any output events. We extend the simple 2 LP scenario to the general case via the use of rb-messages.

All events in the ICEQs and the ICPQs are maintained in receive-time chronological order. From observations 1 and 2 and Rule 1, all events are also in send-time chronological order. The event which has the smallest receive-time in an event queue is referred to as the head event and the event which has the largest receive-time in an event queue is referred to as the tail event. We do not distinguish between messages and events in the rest of the paper.

In Time Warp, an event causing rollback is called a *straggler*. After receiving a straggler, an LP must be rolled back to a previous time point. We call the time which the LP is rolled back to the *rollback-time* and call the first event executed by an LP after rolling back a *rollback-event*. The following Propagation Rule is enforced:

- *Rule 4: If a rollback-event is processed, the resulting output events must be propagated.*

The output events which are generated by the rollback-event are called rb-messages in this paper.

When a new event, E_{new}, arrives at an LP, its receive-time is checked. If the receive-time of E_{new} is larger than or equal to LVT it is scheduled as described in Section 2.4. If the receive-time of E_{new} is smaller than LVT, it is a straggler (e.g. E6 at LP1 in Fig. 8). In this case the LP which receives the straggler is rolled back as follows:

(1) Step 1: The rollback-time is set equal to the receive time of E_{new}.
(2) Step 2: The current LP event which is pointed by CLE is moved from CLEQ to the LPEQ.
(3) Step 3: The current input channel events pointed by respective CIEs are moved from LPEQ to the head of respective ICEQs.
(4) Step 4: Every input channel is rolled back. There are two cases to be considered:
   (a) Case 1: The input channel is the one which receives the straggler. This input channel erases all events in its ICEQ if any, and all ICPQ events which have receive-time larger than the rollback-time.
   (b) Case 2: The input channel is not the one receiving the straggler. This input channel is rolled back by moving all ICPQ events which have receive-time larger than the rollback-time from ICPQ to ICEQ.
(5) Step 5: The LP restores the states to the first state that has time-stamp smaller than or equal to the rollback-time.
(6) Step 6: E_{new} is enqueued at the head of the ICEQ.
(7) Step 7: Every input channel submits one event to the LPEQ if its ICEQ is not empty. LP places one event to the CLEQ. This event is the rollback-event.
(8) Step 8: After the rollback-event is processed, according to *Rule 4*, the output events (rb-messages) must be propagated to descendant LPs. There are five cases to consider when an LP receives a rb-message. Fig. 7 depicts these cases while Fig. 8 depicts concrete examples of these five cases.

   In Fig. 8 input channel 1 receives events from LP1 and input channel 2 receives events from LP2. We assume that LP1 has two service times of 1 and 10. If LP1 processes an event at LVT 6 using service time 1, the output event will be E7 with receive-time 7 and send-time 6; if LP1 processes an event at LVT 6 using service time 10, the output event will be E16 with receive-time 16 and send-time 6.

   Each of five cases is handled as follows:
   (a) Case 1: In this case, the rb-message is not a straggler. The input channel which receives the rb-message erases all of the events which have send-time larger than that of the rb-message from its ICEQ. In Fig. 8, rb23 is propagated from LP1 to LP3 after the rollback event e22 is processed with service time 1 at LP1.

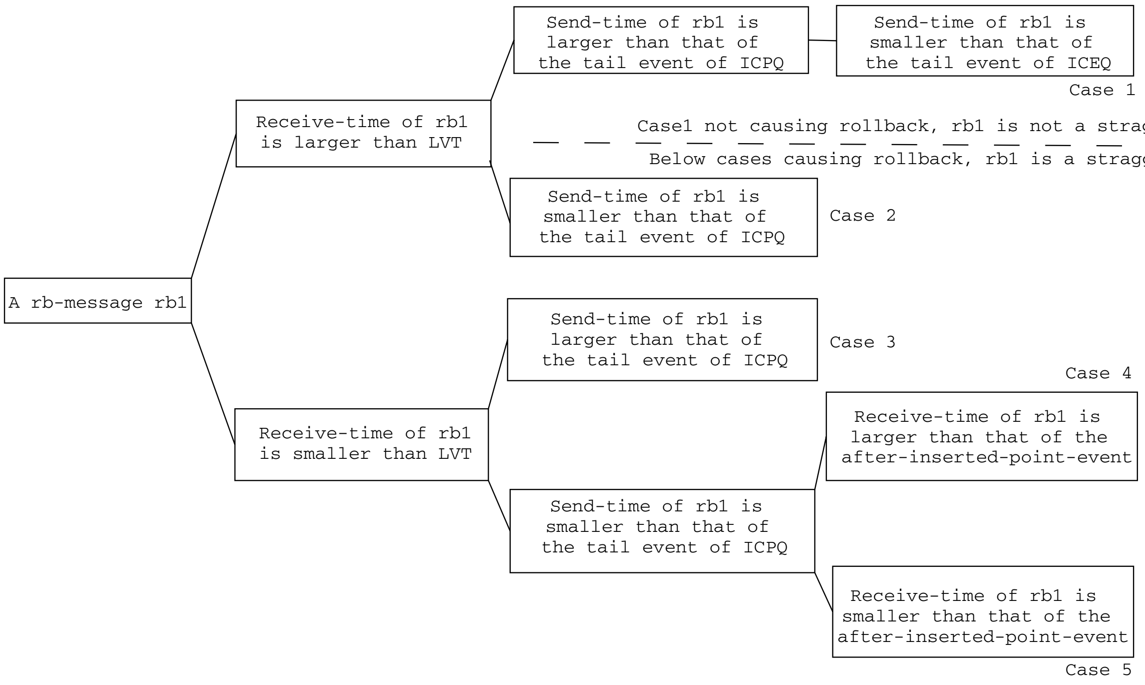
A rb-message rb1
Receive-time of rb1 is larger than LVT
Send-time of rb1 is larger than that of the tail event of ICPQ
Send-time of rb1 is smaller than that of the tail event of ICEQ
Case 1
Case1 not causing rollback, rb1 is not a straggler
Below cases causing rollback, rb1 is a straggler
Send-time of rb1 is smaller than that of the tail event of ICPQ
Case 2
Receive-time of rb1 is smaller than LVT
Send-time of rb1 is larger than that of the tail event of ICPQ
Case 3
Case 4
Receive-time of rb1 is larger than that of the after-inserted-point-event
Send-time of rb1 is smaller than that of the tail event of ICPQ
Receive-time of rb1 is smaller than that of the after-inserted-point-event
Case 5

Fig. 7. Five rb-message arriving cases.

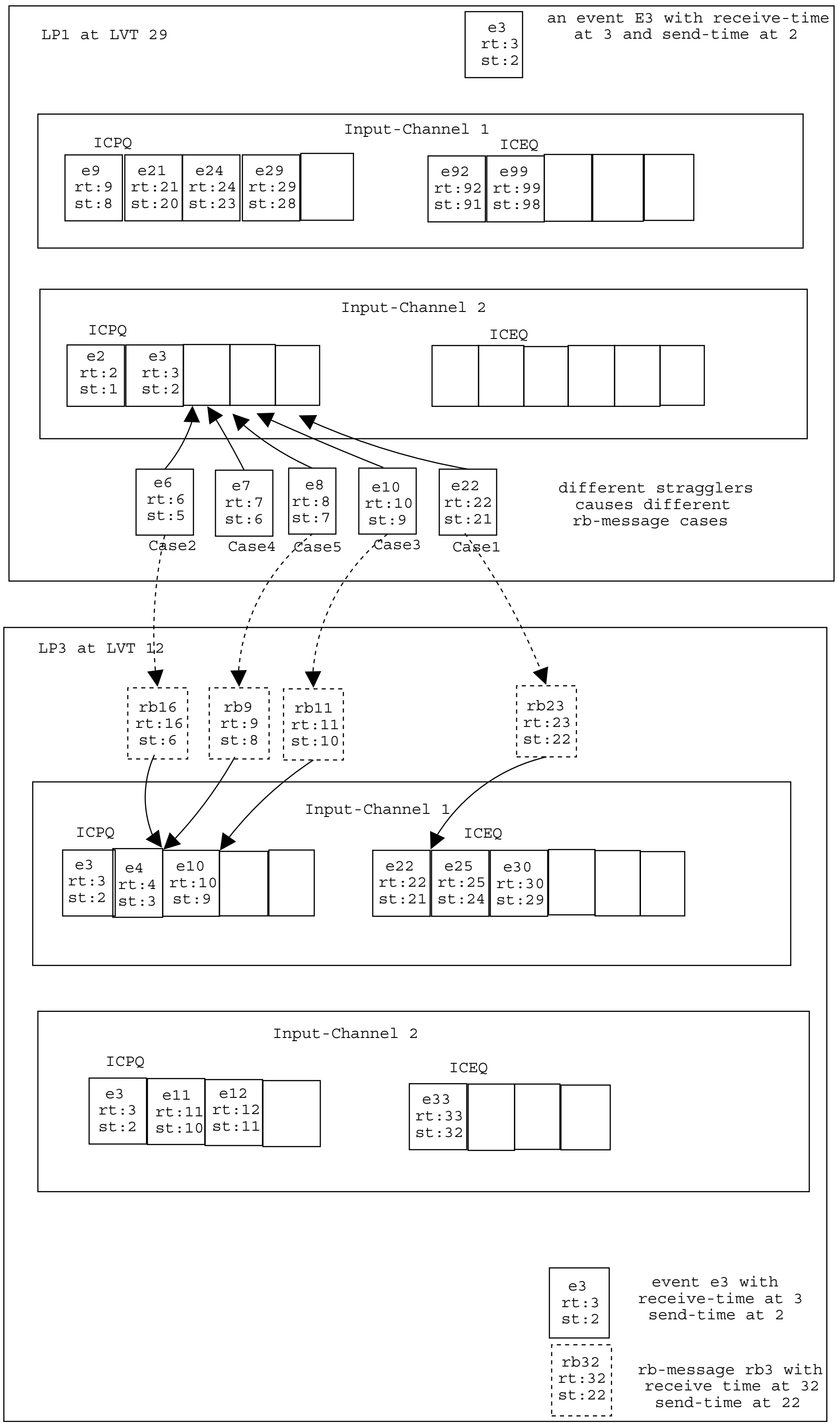

Fig. 8. Concrete examples of five rb-message arriving cases.

(b) Case 2: In this case, the rb-message is a straggler. The send time of the rb-message is used to find the cut-point in the ICPQ, such that all events with a send-time larger than the send-time of the rb-message are after the cut-point. The LP sets the rollback-time equal to the receive-time of the first event after the cut-point. Then the LP recursively applies the rollback procedure described in Steps 2–8 above.

In Fig. 8, rb16 represents this case. Rb16 is propagated from LP1 to LP3 after the rollback event e6 is processed with service time 10 in LP1. Using the send-time 6 of Rb16, the cut-point is found before E10 in ICPQ. The LP3 rollback-time is then set to receive-time 10 of E10. Fig. 9 shows the LP3 after being rolled back by rb16.

(c) Case 3: In this case, the rb-message is a straggler. The LP sets the rollback-time equal to the receive-time of the rb-message. Then the LP recursively applies the rollback procedure following Steps 2–8.

In Fig. 8, rb11 represents this case. Rb11 is propagated from LP1 to LP3 after the rollback event e10 is processed with service time 1 at LP1. In this case, LP3's rollback-time is set to 11 after receiving rb11.

(d) Case 4: In this case, the rb-message is a straggler. The send time of the rb-message is used to find the cut-point in ICPQ. The LP sets the rollback-time equal to the receive-time of the first event after the cut-point. Then the LP recursively applies the rollback procedure following Steps 2–8 as described above.

In Fig. 8, we assume that the following activities happen: (1) E33 in LP3 has been processed and LP3 is at LVT 33. (2) The rollback-event e7 is processed at LP1 with service time 10, and a rb-message, rb17, is generated with receive-time 17 and send-time 7. (3) rb17 is propagated to LP3. When rb17 arrives at input channel1 of LP3, its receive-time (17) is smaller than the LVT (33). Rb17 is a straggler. Using the send-time 7 of rb17, the cut-point is found before E10 which has send-time 9. Since the receive-time 17 of rb17 is larger than that of E10, 10, LP3 sets the rollback-time to the receive-time of E10, i.e. 10.

(e) Case 5: In this case, the rb-message is a straggler. The LP sets the rollback-time equal to the receive-time of the rb-message. Then the LP recursively applies the rollback procedure of Steps 2–8.

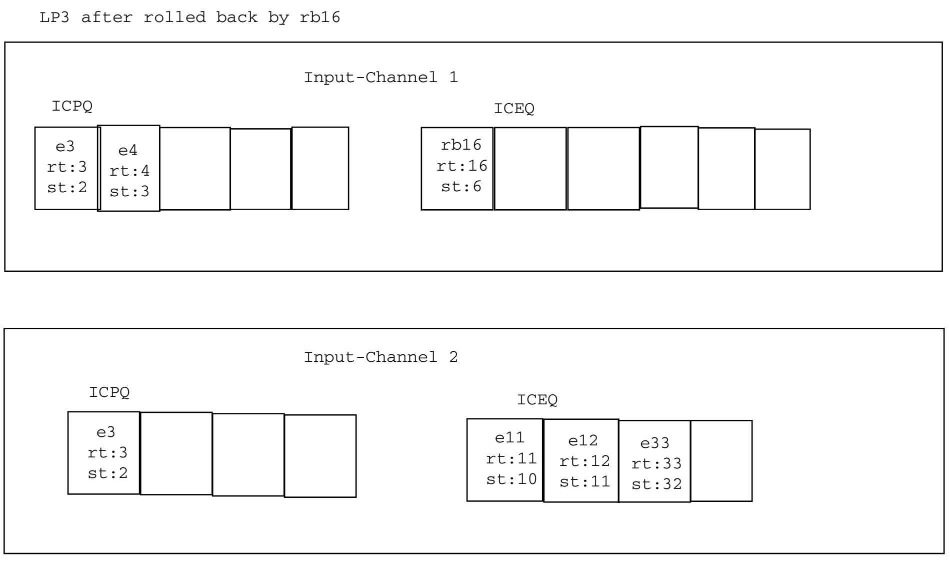

Fig. 9. Rb-messages, an LP receives a straggler rb-message.

In Fig. 8, rb9 represents this case. Rb9 is propagated from LP1 to LP3 after the rollback event e8 is processed with service time 1 at LP1. In this case, the LP3 rollback-time is set to 9 after receiving rb9.

Recursively applying the roll back, send rb-messages procedure will eventually erase all incorrect computations resulting from the original incorrect message send.

From the above description, we can see that the anti-messages mechanism is eliminated in XTW, and therefore the output queue, which is used to store all of the anti-messages can be dispensed with as well. Since an anti-message is saved for every output event, considerable time and space is saved with the elimination of the output queue. This is one of the fundamental virtues of the rb-messages mechanism.

3. Experimental evaluation of XTW

In this section, three sets of experiments are presented. In Section 3.2 a set of experiments is presented to show how much the rb-messages mechanism contributes to simulation performance. In Section 3.3, a set of experiments is described which compares Clustered Time Warp to XTW, while in Section 3.4 a set of experiments examines the scalability of XTW.

3.1. Experimental environment

The XTW–Clustered Time Warp experiments were conducted on a Myrinet network of seven personal computers. Each computer is equipped with dual Pentium III 450 processors and 256 MB of internal memory. The rb-messages mechanism experiments and the scalability experiments were conducted on CLUMEQ [4]. CLUMEQ is a Beowulf cluster with 128 Appro 1100i 1U nodes connected by a Myrinet. Each CLUMEQ node has dual Athlon 1900+ processors with 3 G bytes of memory.

All of the results reported in this paper are the average values of multiple runs. XTW employs MPI, thereby guaranteeing a FIFO order of message communication.

The following metrics are made use of for our performance evaluation:

- *Simulation Time*: defined as the elapsed real time for the simulation. The partitioning time is included in all XTW results.
- *Speedup*: defined as the ratio of the simulation time of a simulator using one processor to the simulation time of the same simulator using more than one processor.
- *Throughput*: defined as the number of processed events per second.
- *Goodput*: defined as the number of committed, processed events per second.
- *Committed rate*: defined as the ratio of the *Goodput* to the *Throughput*.

3.2. Rb-messages mechanism

In XTW, the rb-messages mechanism replaces the anti-message mechanism. In order to determine how much the rb-messages mechanism alone contributes to the simulation performance, we created a simulator which is identical to XTW except for its use of anti-messages, and compared its performance to that of rb-messages. XEQ is used in both simulators along with the same event scheduling mechanism. The *Simulation Time Ratio*, defined as the ratio of the simulation time using anti-messages to the simulation time using rb-messages, is the metric used for this comparison.

Fig. 10 shows that XTW using rb-messages has a 1.2–2.3 relative speedup compared to XTW using anti-messages. In Fig. 10, the 10M circuit *simulation time ratio* jumps from 1.3 with 12 processors to 2.3 with 16 processors. We noted that with 12 processors, both the rb-message simulator and the anti-message simulator exhibit considerable memory swapping. With 16 processors, because the rb-message simulator eliminates the

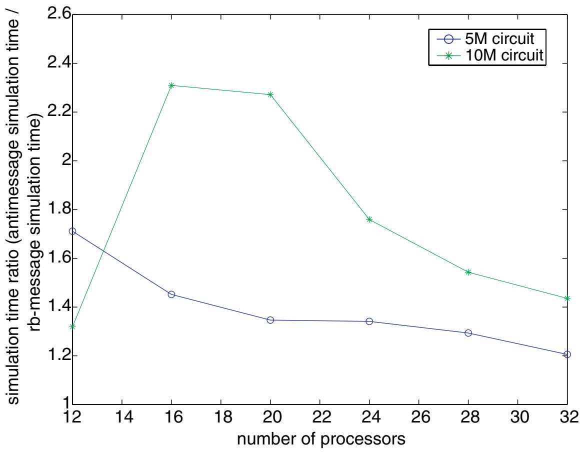

Fig. 10. Simulation time ratio (anti-message vs. rb-message).

output queue it consumes less memory—a negligible amount of memory swapping was observed. Nevertheless, the anti-message simulator with 16 processors has the same amount of memory swapping as with 12 processors. Thus the simulation time ratio jumps to 2.3 with 16 processors. When more processors are used, memory swapping is reduced in both the rb-message simulator and the anti-message simulator, and the simulation time ratio is dominated by the efficiency difference between the mechanisms.

3.3. XTW vs. Clustered Time Warp

In this section, we present results comparing the performance of XTW and Clustered Time Warp [3,5]. We make this comparison because Clustered Time Warp is oriented toward logic simulation, and exhibits a superior performance to Time Warp [3] in this domain.

In our experiments, the *local roll back*, *local checkpoint* mechanism is made use of in Clustered Time Warp. Local checkpoint means that an LP saves its state only if it receives a message from an LP in another cluster. Local rollback refers to each LP rolling back individually, i.e. the same technique used in Time Warp.

We conducted experiments on various benchmark circuits. The results show that Clustered Time Warp has the best performance on circuit s90k—a combination circuit which consists of ISCAS89 benchmark circuits s38584 and s38417, and has around 90,000 gates. We present the XTW–Clustered Time Warp comparisons making use of s90k. Both Clustered Time Warp and XTW use the same partitioning algorithm. The time to perform the partitioning is not included in the simulation time.

Since Clustered Time Warp crashes when more than 3 processors are used in a simulation, all of the Clustered Time Warp results are presented with up to 3 processors.

3.3.1. Simulation time

Fig. 11 shows the simulation time vs. the number of processors. The results demonstrate that XTW outperforms Clustered Time Warp in all parallel simulations with any number of processors.

3.3.2. Goodput and committed rate

Fig. 12 depicts the goodput vs. the number of processors. Fig. 13 shows the committed rate vs. the number of processors. Fig. 12 shows that XTW has an almost linear increase in the goodput, while Clustered Time Warp has a relatively flat one. Fig. 13 reveals the reason behind this phenomenon—XTW has a higher committed event rate than Clustered Time Warp. Moreover, XTW has an almost flat reduction in the committed event rate when more processors are used, while Clustered Time Warp has a relatively steep reduction in its committed event rate. *These results indicate that XTW has a more efficient rollback mechanism.*

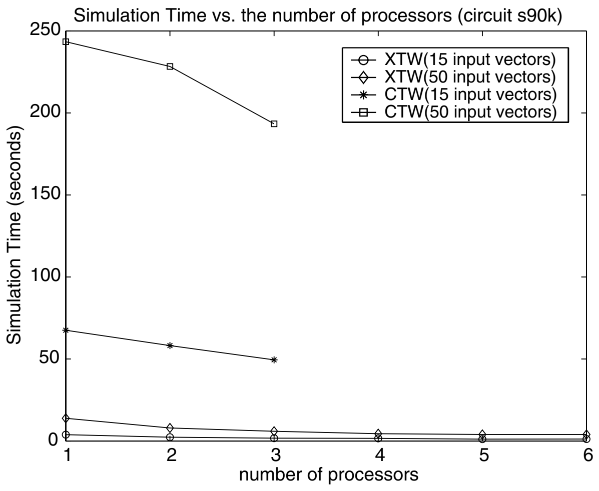

Fig. 11. Simulation time vs. number of processors.

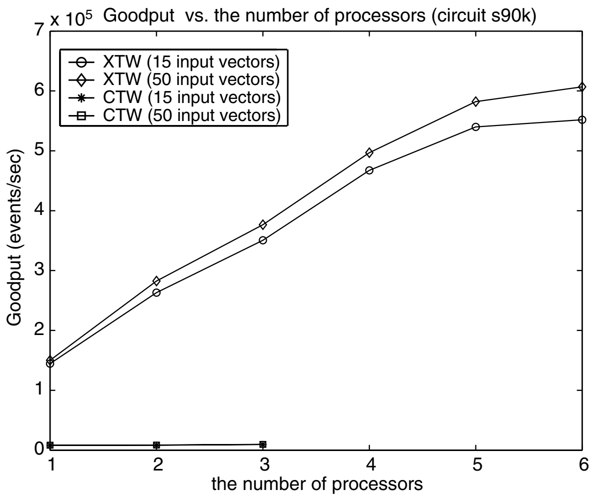

Fig. 12. Goodput vs. the number of processors.

3.3.3. Speedup

Fig. 14 shows speedup vs. the number of processors. Although XTW has a much larger goodput than Clustered Time Warp, our results indicate that XTW has a larger speedup than Clustered Time Warp in all the cases. Moreover, XTW has an almost linear increase in speedup while Clustered Time Warp has a relatively flat one. This clearly demonstrates that XTW has a smaller overhead than Clustered Time Warp.

3.4. XTW scalability experiments

In this section we explore the scalability of XTW with respect to the size of the circuit and the number of processors. Three circuits (5-million-gate, 10-million-gate and 25-million-gate) are simulated on CLUMEQ.

There is an absence of large benchmark circuits described as gate-level netlists in the public domain. Consequently, we developed a hierarchical mechanism to generate the synthetic circuits which we used in the experiments described in this section.

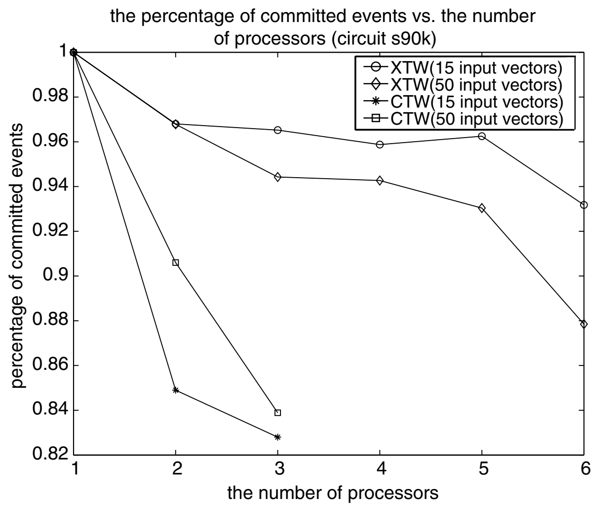

Fig. 13. Committed events rate vs. the number of processors.

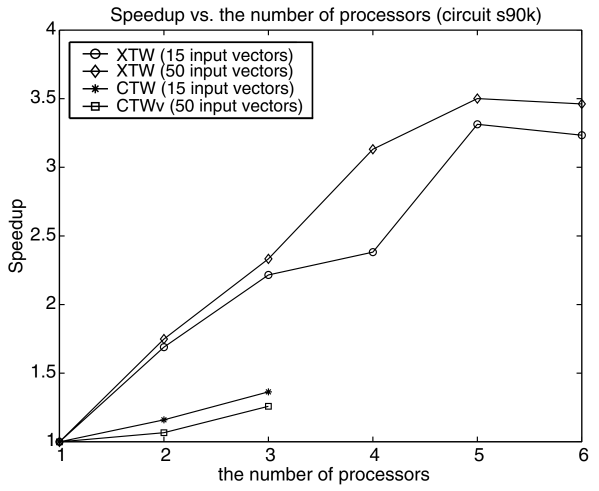

Fig. 14. Speedup vs. the number of processors.

The benchmark circuits are generated as follows:

(1) A module-level netlist, consisting of real world circuits (which can be described in Verilog or VHDL) is created. This module-level netlist is used to describe the connections between the modules of the benchmark circuit.
(2) Each node (module) of the module-level netlist is instantiated into a gate-level netlist.

By making use of this hierarchical approach, we can generate a synthetic benchmark circuit of any size. The design of a large circuit follows a divide and conquer approach, in which the design is broken up into a collection of individual modules and in which the interfaces between individual modules are clearly defined. As a consequence of this approach, most communication occurs within (as opposed to between) the modules. Based upon this observation, we made use of a straightforward DFS-module partitioning algorithm in our experiments-the algorithm partitions the modules of the circuit design. One shortcoming of this approach is that it can result in an unbalanced partition, i.e. different numbers of gates can be assigned to different computers.

Table 1
Goodput (events/s) vs. number of processor

Number of nodes	5-Million-gate	10-Million-gate	25-Million-gate
58	7979605.22	7409170.01	5127921.82
52	7398898.55	6640353.16	4224213.26
48	6076230.10	6116893.97	3073751.46
44	5642259.97	5792584.54	2596966.32
40	5409749.92	4753970.33	1890728.53
36	4823315.03	4233274.93	1442845.16
32	4214269.92	3581280.84	1223729.46
28	3988396.51	3031265.69	1038491.47
24	3197840.34	2194713.49	848558.51
20	2503215.12	1642900.35	1005911.43
16	1719632.75	891090.69	688868.01
12	1227617.98	454356.90	N/A

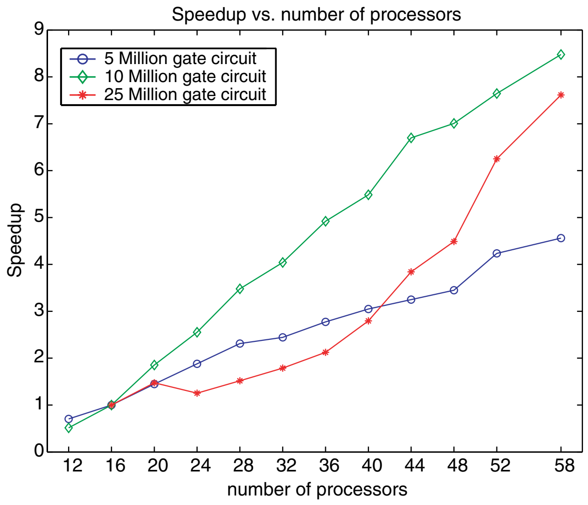

Fig. 15. Speedup vs. the number of processors.

Table 1 shows the goodput of three circuits simulated on 12–58 CLUMEQ nodes. The simulations of the 25-million-gate circuit on 12 nodes could not be completed. From the data in Table 1, we see a consistent trend of an increasing goodput with the number of processors. The goodput decreases as the size of circuit increases, a consequence of a larger number of events in a larger circuit imposing a heavier load on the communications network and the memory hierarchy.

Since the 25-million gates circuit can only be simulated with 16 or more processors, we use the simulation time with 16 processors as the base with which to calculate the speedups. Fig. 15 shows the speedup vs. the number of processors. In Fig. 15, we see that XTW scales almost linearly with the number of processors and scales well with the size of the circuits.

4. Related work

In append-queues [6], the events from a sender LP are appended to a sender-queue associated with its Id at the receiver. A schedule-list which sorts the head element of each sender queue is maintained for event scheduling. An event that has been scheduled and processed is appended to a processed-queue associated with the event's receiver Id. Aggressive cancellation is used in append-queues. Append-queues was compared with a multi-list algorithm using the WARPED [7] simulation kernel. Append-queues was observed to have a lower cost for normal event scheduling and a higher cost for rollback event scheduling. The extra scheduling cost for rollback happens in event re-insertion. In append-queues, processed-queues have no relationship to sender-queues, so each rolled back event has to be re-inserted one by one from a processed-queue back to a respective sender-queue. This causes an extra scheduling cost compared to the multi-list algorithm. In the XEQ event-list structure, a new event is appended to an event queue in an input channel. After the event is processed, it is appended to the processed-queue in the same input channel (Fig. 4). When rollback happens in XTW, only one search operation is needed in each processed-event queue to find the cut-point. The event queue is appended to the end of processed-queue. The event after the cut-point becomes the head element of the event queue (Section 2.1). Thus, the XEQ structure does not have the extra scheduling cost in rollback which is a part of append-queues. Moreover, the new rb-message mechanism in XTW further reduces the rollback cost.

5. Conclusion

In this paper, two new mechanisms for improving the efficiency of distributed logic simulation were introduced. The first, XEQ, is a multi-level input queue which lies behind an O(1) event scheduling algorithm. The second, rb-messages, reduces the rollback costs. It also reduces the cost of saving events by eliminating the output queue at each LP. Both of these mechanisms assume the use of clusters of LPs. These mechanisms were combined with a version of Clustered Time Warp to produce a simulation framework which we call XTW.

The cost of these algorithms was analyzed in theory. Comparisons to Clustered Time Warp revealed that XTW has a far superior performance. Clustered Time Warp out-performs Time Warp in logic simulation. Experimental comparisons to a sequential version of XTW suggested that it is scalable, while experiments with large, synthetic circuits further support this conclusion.

It is certainly desirable to make use of XTW on large, real circuits and to modify it for use in behavioral and mixed behavioral/logic simulations. In addition, the development of efficient partitioning and/or load balancing algorithms is vital for the further development of XTW. We hope to continue our work in these directions.

Appendix A. Optimization techniques for parallel logic simulation

Many optimization techniques for Time Warp have been developed in order to attack different overheads, to stabilize Time Warp or to simply add useful features. We describe four of these techniques which are made use of by XTW:

(1) Rollback Relaxation
(2) Clustered Time Warp
(3) Bounded Time Window
(4) Event-lookahead Time Warp.

A.1. Rollback relaxation

Rollback relaxation is a novel technique for attacking the state-saving overhead in Time Warp [8]. To apply *rollback relaxation*, LPs are classified into Time Warp categories: *memoryless* and *memoried* LPs. A memoried LP is actually an ordinary LP in Time Warp. The output of a memoried LP is a function of both input values and internal state values. In such LPs event processing may use internal state information from previous event processing activities in order to produce an output event. Thus a state-saving mechanism must be implemented in a memoried LP in order to enable the restoration of state variables in case of a rollback. A memoryless LP's output behavior is completely determined by the values of its inputs. Event processing by a memoryless LP will never use internal state information from past event processing to produce an output event.

All memoryless LPs qualify for *rollback relaxation*. In rollback relaxation, no state is saved during processing. When a straggler arrives, the LP reconstructs any required input state from the events of input queues. Wilsey et al. proposed the use of multiple input queues in order to accelerate the search on each input variable for state reconstruction.

In logic simulation, the gates (AND, OR and XOR etc.) can be modeled as memoryless LPs. Obviously, the *rollback relaxation* mechanism can reduce the state-saving overhead by a considerable amount if there are a large number of memoryless LPs in the simulation.

A.2. Clustered Time Warp

Clustered Time Warp (CTW) is a hybrid system in which LPs are scheduled sequentially within clusters, and clusters are synchronized by Time Warp [9]. Clustered Time Warp has the following three variations:

- Clustered Rollback–Clustered Checkpoint (CRCC): In this technique, all of the LPs in a cluster are rolled back upon the arrival of a straggler. This is the most conservative approach and requires less memory than the Time Warp other checkpointing techniques. Because all of the processes in a cluster are rolled back, the performance of the simulation is decreased.
- Local Rollback–Local Checkpoint (LRLC): Individual LPs perform their own rollbacks as in Time Warp. Checkpoints are taken upon the arrival of a message from another cluster. This is the closest of the techniques to a pure Time Warp and performs fairly well in terms of execution time. However, the price to pay is memory.
- Local Rollback–Clustered Checkpoint (LRCC): This is the midway between CRCC and LRLC and gives performance results between CRCC and LRLC in terms of both execution time and memory consumption.

Experimental results [9] indicate that the LRLC approach is the fastest and that it consumes more memory then the other approaches. Since we apply other techniques to reduce memory usage, we make use of the LRLC approach in order to minimize the simulation time.

A.3. Event-lookahead Time Warp

The *Event-lookahead Time Warp* (ETW) [10] technique reduces unnecessary intermediate events by combining multiple input events which arrive at each gate within the same clock cycle and generating one output event for these combined events. This is done instead of generating individual output event(s) for each input event. Clearly, more events executing in the same clock-cycle results in a better efficiency. However, in unit-delay logic simulation, a one unit time lookahead is too small to engage large numbers of events within each clock-cycle. Thus, ETW has a limited effect on unit-delay logic simulations. It should be noted that all of the experiments described in this paper are unit-delay simulations.

A.4. Bounded Time Window

It is possible for an overly optimistic LP or cluster to be rolled back and thereby cause many of its descendants to roll back as well. Moreover, this increased number of rollbacks may cause Time Warp to be unstable. A simple approach to preventing some LPs from advancing too far ahead of the pack in is to bound how far one LP can advance ahead of the others in virtual time. The *Bounded Time Window* (BTW) mechanism is an example of this approach [11,12].

A variation of this approach is to define the window in terms of the number of processed, uncommitted events (NPUE) that may reside at an LP. In Breathing Time Warp [13], the user must specify this NPUE parameter. An LP is blocked when the number of processed events at the LP with time-stamp larger than GVT reaches NPUE. The LP becomes unblocked when the GVT advances and some of these events are committed.

References

[1] Q. Xu, C. Tropper, XTW, a parallel and distributed logic simulator, in: Principles of Advanced and Distributed Simulation, PADS 2005, Monterey, California, Workshop on 01–03 June, 2005, pp. 181–188.
[2] D. Jefferson, Virtual time, Program. Languages Syst. 7 (3) (1985) 404–425.

[3] H. Avril, C. Tropper, On rolling back and checkpointing in time warp, IEEE Trans. Par. Distr. Syst. 12 (11) (2001) 1105–1122, Nov. 2001.

[4] Clumeq infrastructure, 2003. Available from: <www.clumeq.mcgill.ca>.

[5] H. Avril, C. Tropper, Scalable clustered time warp and logic simulation, VLSI Design, Special Issue on Current Advances in Logic Simulation, Gordon-Breach 19 (3) (1999) 291–313.

[6] J. Dahl, M. Chetlur, P.A. Wilsey, Event list management in distributed simulation, in: Lecture Notes in Computer Science, vol. 2150/2001, Springer-Verlag GmbH, 2001, p. 466.

[7] R. Radhakrishnan, D. Martin, M. Chetlur, D. Rao, P. Wilsey, An object-oriented time warp simulation kernel, in: Lecture Notes in Computer Science, vol. 1505, Springer-Verlag, London, UK, 1998, pp. 13–23.

[8] P. Wilsey, A. Palaniswamy, Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation, in: International Conference on Simulation and Hardware Description Languages (January 1994), Society for Computer Simulation, 1994, pp. 143–148.

[9] H. Avril, Clustered time warp and logic simulation, Ph.D. dissertation, McGill University, 1996.

[10] H. Kim, J. Jean, Parallel optimistic logic simulation with event lookahead, in: International Conference on Parallel Processing, 10–14 August, 1998, pp. 20–27.

[11] A.C. Palaniswamy, Dynamic parameter adjustment to speedup time warp simulation, Ph.D. dissertation, University of Cincinnati, 1994.

[12] S. Turner, M. Xu, Performance evaluation of the bounded time warp algorithm, in: 6th Workshop on Parallel and Distributed Simulation, 1992, pp. 117–126.

[13] J. Steinman, Breathing time warp, in: 7th Workshop on Parallel and Distributed Simulation, 1993, pp. 109–118.