XTW, a parallel and distributed logic simulator

Qing XU
School of Computer Science
McGill University
Montreal, Quebec H3A 2A7
Email: qxu2@cs.mcgill.ca

Carl Tropper
School of Computer Science
McGill University
Montreal, Quebec H3A 2A7
Email: carl@cs.mcgill.ca

Abstract— **In this paper, a new event scheduling mechanism XEQ and a new rollback procedure rb-messages are proposed for use in optimistic logic simulation. We incorporate both of these techniques in a simulator XTW. 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 Time Warp reveal a superior performance on the part of XTW, while experimental results over large circuits (5-million-gate to 25-million-gate) shows XTW scales well with both the size of circuits and the number of processors.**

1 Introduction

In the competitive arena of VLSI design, the size of circuits has increased as Moore's law predicted -the transistor density on integrated circuits doubles every couple of years. One result of this is the steadily increasing computational requirements for circuit simulation and verification. Parallel and distributed simulation has the potential to provide a solution to this problem.

The fundamental problem in distributed simulation is one of synchronizing the processes involved in the simulation. The two major approaches to synchronization are referred to as conservative and optimistic. We focus upon the optimistic algorithms in this paper, of which Time Warp [1] is the most visible. In Time Warp (TW) 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.

Logic and behavioral simulation is a low granularity and tightly coupled computational task which poses a significant challenge to the development of a distributed simulator. Clustered Time Warp (CTW) [2], [3] was developed with these problems in mind. As the name implies, in CTW LPs (representing gates) are gathered into clusters. Several techniques were developed for use to obtain checkpoints and to roll back the LPs in a cluster. The algorithms described in this paper are intended for use with clusters of gates and are an outgrowth of CTW.

A number of other efforts have been directed at logic simulation including [4] which employs optimistic algorithms and [5], which employs conservative algorithms. [5] contains a good survey of work before 1995.

In this paper, a new optimistic synchronization mechanism, XTW, is proposed to improve the performance of Time Warp. XTW was inspired by several characteristics of discrete event logic simulation. XTW consists of a new event scheduling algorithm, *XEQ*, and a new rollback mechanism, *rb-message*. XEQ has an O(1) cost bounded on the number of simulated entities (not on the number of events). *Rb-message* not only reduces the computing cost of annihilating previously sent messages, but also dramatically reduces the memory cost by eliminating the output queue in each LP.

The remaining sections of the paper are organized as follows. Section2 describes the motivation for XTW. Section 3 contains a detailed description of XEQ and rb-messages along with an analysis of their complexity. Section 4 contains our experimental work, in which XTW is compared to CTW as well as to a sequential version of XTW. The concluding section of the paper follows.

2 Motivation for XTW

Discrete event simulations of circuits, whether at the logic, behavioral or register-transfer level, share certain characteristics, among which are:

1) Events generated by an LP are produced in chronological order(See figure 1).
2) An LP receives events from another LP in chronological order(See figure 1).
3) In a parallel discrete event simulation which uses a communication facility guaranteeing FIFO order, the messages(events) from a specific LP(source) arrive at the destination LP in chronological order.
4) LPs are *sparsely connected*.
5) The LP topology is static during the simulation.

Observations 1, 2 and 3 show that events are naturally sorted with "zero-cost" when they are generated and propagated. These observations are the keys to our approach and inspire us to create a new event scheduling algorithm, XEQ, which utilizes these "zero-cost" sorted events. We make use of XEQ to create the *rb-message* mechanism in order to reduce the rollback cost in TW. Observations 4 and 5 make it feasible to implement XEQ and rb-message in large circuit simulations.

2.1 Utilizing "zero-cost" sorted events

In this section, we will explore how to utilize "zero-cost" sorted events in discrete event circuit simulation.

First, we examine a simple situation– a parallel discrete event circuit logic simulation which has two components residing on two simulation nodes. One component is an event-generator and resides on Node1, while another component is a NOT gate residing on Node2. Each component is modeled as an LP and communicates with each other via FIFO order communication facility.(See figure 1). In figure 1, we can see

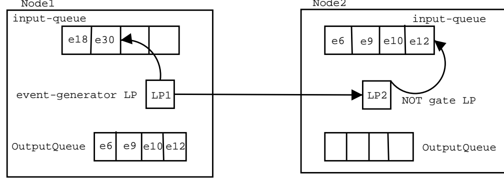

Fig. 1. a single LP, single InCh model in PDES

that the event scheduling cost at Node1 is O(1), consisting of the cost to append the generated events into the input queue and to de-queue the head event from input-queue. At Node2, the event scheduling cost is also O(1), consisting of the cost to append the events coming from Node1 to the input-queue and to de-queue the head event from the input-queue. In this example, observations 1, 2 and 3 are made use of to give an O(1) cost event scheduling algorithm.

When there are a large number of LPs residing in one node and multiple input sources at one LP, the situation is totally different. Events generated by different LPs and coming from different sources have to compete with other to be inserted into the input-queue.(see figure 2) The naturally occurring “zero-cost” sorted events are lost by this competition. In TW, 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-message, is proposed to make it possible to utilize “zero-cost” events to reduce the rollback overhead. We call the Time-Warp simulation system which implements XEQ and rb-messages XTW.

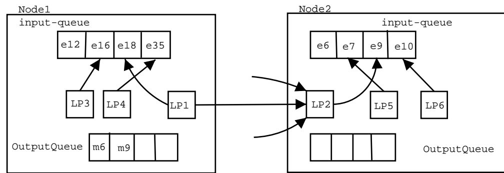

Fig. 2. a multiple LPs, multiple InChs model in PDES

3 XTW

XTW makes use of clusters of LPs. The clusters are intended to represent the grouping of gates according to the functional units to which they belong. Each cluster has a multi-level event queue, *XEQ*, associated with it whose purpose is event scheduling. A cluster level event queue (*CLEQ*) is part of XEQ and stores events which are sent to other clusters. XTW is an outgrowth of CTW [2]. Three techniques for checkpointing and rolling back in CTW are described in [2], 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 all of the LPs in a cluster to roll back (one of the techniques in CTW).

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 3.1 introduces the *Input-Channel* structure. Section 3.2 presents the structure of *XEQ*. Section 3.4 presents the XTW event scheduling mechanism and its cost analysis. Section 3.5 presents the *rb-messages* mechanism.

3.1 Input-Channel

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

Rule 1: Each InCh can only have one unique incoming source.

Figure 3 shows how the *Input-Channel* models the connection edge of gates. In figure 3, G1 has two inputs from G2 and G3. G2 has one input. G3 has one input from itself and another from others. Each input is modeled as an InCh.

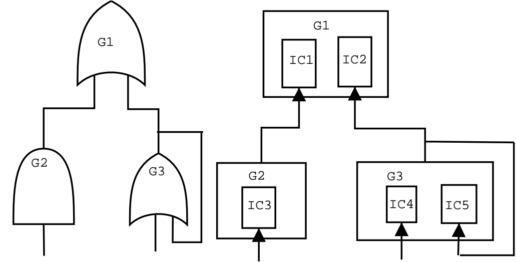

Fig. 3. Input Channel Model

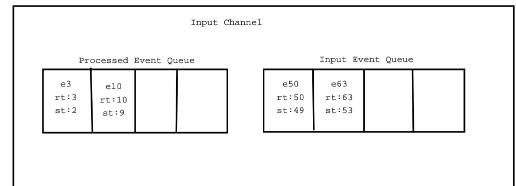

Fig. 4. The Structure of Input Channel

Figure 4 shows the structure of InCh. Each InCh 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 two timestamps: a)receive-time stamp is the time stamp indicating when the event occurs(conventional definition of time stamp) b)the send-time stamp is the Local Virtual Time(LVT) of the LP when it scheduled E . LVT is equal to the receive-time of the latest processed event.

As a result of observations 1, 2, and 3 (the *FIFO communication assumption*) and *Rule 1*, all of the events must arrive at each ICEQ in chronological order and be naturally sorted in the ICEQs(see Figure 4).

3.2 The Structure of XEQ

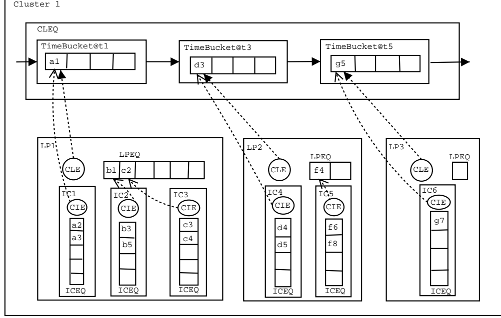

Fig. 5. The Structure of XEQ

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

In addition, the following two 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.

3.2.1 Rules for XEQ: The following rules are enforced in XEQ:

- *Rule 2: An InCh 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.*

3.3 Event Node Structure, Space Cost of XEQ

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

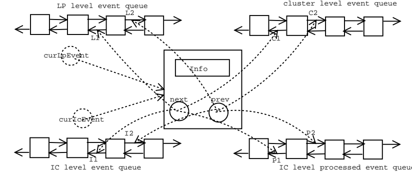

Fig. 6. an event node structure and its movement

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 figure 6. When e1 is moved from the ICEQ to the LPEQ, the only operation necessary is changing the next and prev pointer of e1 from I1,I2 to L1, L2. Similarly, moving e1 to the CLEQ or ICPQ just involves changing the next and prev pointer value 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.

3.4 XTW O(1) Event Scheduling Mechanism

XEQ is used to implement a smallest timestamp 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 InCh and is appended into the respective ICEQ.
2) According to *Rule 2*, if the ICEQ is not empty, it will submit the smallest receive-time event to its LPEQ. Since the ICEQ is naturally sorted, the smallest timestamp event is just the head event of ICEQ. Thus, we can simply de-queue the head event at a cost of 1.
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 LPEQ. Based on *Rule 2*, in the worst case, the maximum value of N_e is C_{ic}, where C_{ic} is the number of InChs at an LP.
4) According to *Rule 3*, if the LPEQ is not empty, it will submit the head event to its 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. Based on *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 :

$$SC = 1 + C_{ic} + C_{lp} \quad (2)$$

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(lgN) data structure in the CLEQ, results in an event scheduling cost of $O(\log C_{lp})$.

Comparing XEQ 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($O(\log n)$)r. 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.

3.5 Rollback with Rb-messages

3.5.1 Motivation for the Rb-messages *Mechanism:* We begin with the 2-LP example shown in figure 1. This time, we assume that a rollback occurs in LP1- event e8 is generated after e12 in LP1 and is sent to LP2. In Time-Warp, anti-messages for e9, e10 and e12 will be sent out one by one 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 previous sent messages.

The advantage of above scenario is obvious – we can not only can 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*, described in the following section.

3.5.2 The Rb-messages *Mechanism :* In XTW, each event has two timestamps: a)receive-time stamp is the time stamp indicating when the event occurs(conventional definition of time stamp) b)the send-time stamp is the Local Virtual Time(LVT) of the LP when it scheduled E. All events in ICEQs and ICPQs are maintained in receive-time chronological order. According to Observations 1, 2 and 3, all events are also in the send-time chronological order. The event which has the smallest receive-time in an event-queue is called as the head event. The event which has the largest receive-time in an event-queue is called 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 point that the LP is rolled back to as the *rollback-time* and call the first event submitted by an LP after rolling back as a *rollback event*. In XTW the following "Propagation Rule" is enforced in addition to the normal propagation rule:

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

The output-events, which are generated by the *rollback event* and forced to propagate, are mainly used to propagate the rollback and thus called as *rb-messages* in this paper.

In the following, we describe how the *rb-messages* mechanism works in the XTW rollback procedure:

When a new event, E_{new}, arrives at an LP, it is checked if its receive-time is smaller than LVT. If the receive-time of E_{new} is larger than or equal to LVT, it is a normal event and will be scheduled in the way described in section 3.4. If the receive-time of E_{new} is smaller than LVT, it is a straggler(e.g. E6 at LP1 in fig 8). Then, 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 InCh 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 en-queued at the head of its arriving ICEQ.
7) Step 7: Every *input-channel* submits one event to LPEQ if its ICEQ is not empty. LP submits one event to 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 propagate to subordinate LPs. There are five cases to consider when an LP receives a rb-message. Fig 7 states all five cases. Fig 8 depicts concrete examples of these five cases.
   In fig 8, LP3 has two *input-channels*, InCh1 receives events from LP1, InCh2 receives events from LP2. We assume LP1 has two different service-time, 1 and 10. So if LP1 processes an event at LVT 6 using service time 1, the output event will be E7 with receive-time at 7(rt:7) and send-time at 6(st:6); if LP1 processes an event at LVT 6 using service time 10, the output event will be E16 with receiver-time at 16(rt:16) and send-time at 6(st:6).

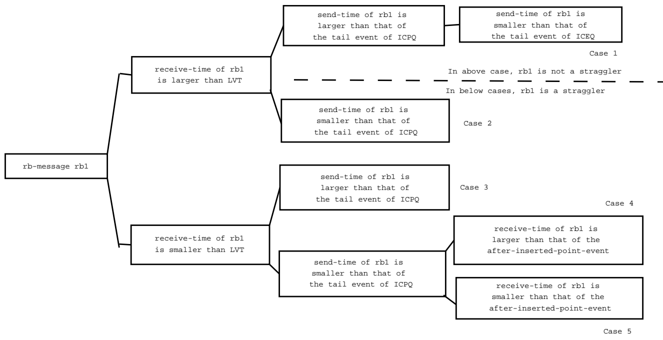

Fig. 7. five rb-message arriving cases

Each of five cases is handled respectively 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 events which have send-time larger than that of the rb-message from its ICEQ. In fig 8, rb23 is in this case. Rb23 is propagated from LP1 to LP3 after the *rollback event* e23 is processed with service time 1 in LP1.

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 ICPQ, such that all events with send-time larger than the send-time of the rb-message is 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 recusively applies the rollback procedure following Step2-Step8 as described above.
In fig 8, rb16 is in 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 of Rb16, st:6, the *cut-point* is found before E10 in ICPQ. The LP3 *rollback-time* is then set to the receive-time of E10 at 10. Fig 9 shows the LP3 after 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 recusively applies the rollback procedure following Step2-Step8 as described above.
In fig 8, rb11 is in this case. Rb11 is propagated from LP1 to LP3 after the *rollback event* e10 is processed with service time 1 in LP1. In this case, the LP3 *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 recusively applies the rollback procedure following Step2-Step8 as described above.

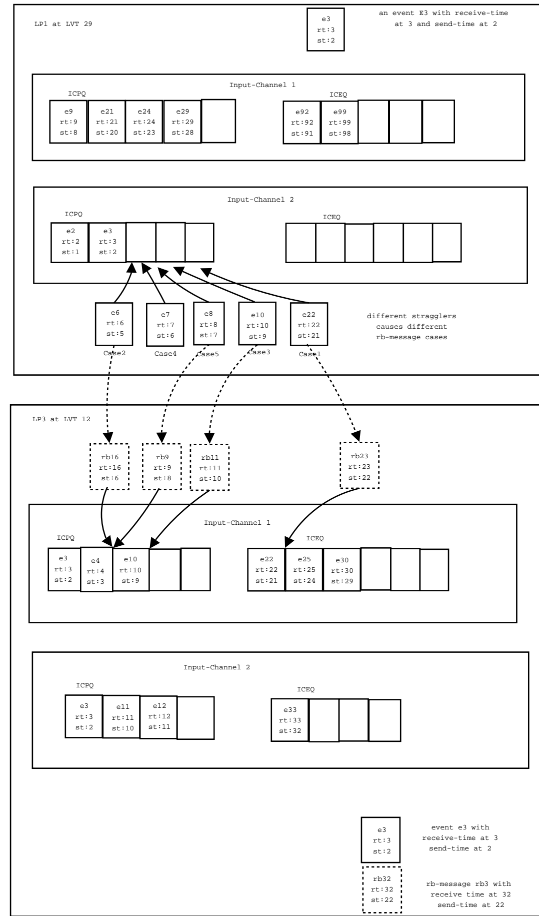

Fig. 8. concrete examples of five rb-message arriving cases

In fig 8, we assume following activities happened: 1) E33 in LP3 has been processed and LP3 is at LVT 33. 2) The *rollback-event* e7 is processed in LP1 with service time 10, and a rb-message, rb17, is generated with receive-time at 17 and send-time at 7. 3) rb17 is propagated to LP3.
When rb17 arrives at InCh1 of LP3, its receive-time(rt:17) is smaller than the LVT(33). Rb17 is a straggler. Using the send-time of rb17, st:7, the *cut-point* is found before E10 which has the send-time at 9. Since the receive-time of rb17(rt:17) is larger than that of E10(rt:10), LP3 sets the *rollback-time* to the receive-time of E10, at 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 recusively applies the rollback procedure following Step2-Step8 as described above.
In fig 8, rb9 is in this case. Rb9 is propagated from LP1 to LP3 after the *rollback event* e8 is processed with service time 1 in LP1. In this case, the LP3 *rollback-time* is set to 9 after receiving rb9.

Recursively applying the above “roll back, send rb-

messages" procedure will eventually erase all incorrect computations resulting from the original incorrect message send.

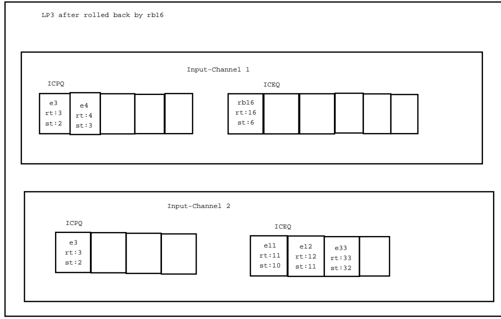

Fig. 9. *rb-messages*, an LP receives a straggler rb-message

3.5.3 Eliminating the Output Queue *and the* anti-messages*:* 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 obviated in XTW as well. Since an anti-message is saved for every output event, considerable time and space are expected to be saved with the elimination of the *output queue*. This is one of the fundamental virtues of the *rb-messages* mechanism.

4 Experimental Evaluation of XTW

In this section, two sets of experiments are presented. In subsection 4.1, a set of experiments is described which compares CTW and XTW, while in subsection 4.2, a set of experiments examines the scalability of XTW.

All of the results reported in this paper are the average values of 10-100 runs. XTW employs MPI as the software communication platform, thereby guaranteeing a FIFO order of message communication.

4.1 XTW vs. Time Warp(CTW)

In this subsection, we present results comparing the performance of XTW and CTW [2] [3]. We compare XTW to CTW because CTW is oriented towards logic simulation, and exhibits a superior performance to Time Warp [2] in this domain.

The XTW-CTW experiments were conducted on a Myrinet network of seven personal computers. Each computer is equipped with dual Pentium III 450 processors and 256 Megabytes of internal memory.

In our experiments, the *local roll back, local checkpoint* mechanism is made use of in CTW. 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 CTW has the best performance on the circuit s90k – a combination circuit which consists of ISCAS89 benchmark circuits s38584 and s38417, and has around 90,000 gates. In the following, we present the XTW-CTW comparisons making use of s90k.

The following metrics are used for the performance comparison:

- *Simulation Time*: *Simulation Time* is defined as the elapsed real time for the simulation.
- *Speedup*: *Speedup* is 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*: *Throughput* is defined as the number of processed events per second.
- *Goodput*: *Goodput* is defined as the number of committed processed events per second.
- *Committed Rate*: *Committed Rate* is defined as the ratio of the *Goodput* and the *Throughput*.

Both CTW and XTW use the same partitioning algorithm. The time to perform the partitioning is not included in the simulation time.

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

4.1.1 Simulation Time: Figure 10 shows *the simulation time vs. the number of processors*. The results demonstrate that XTW outperforms CTW in all parallel simulations with any number of processors.

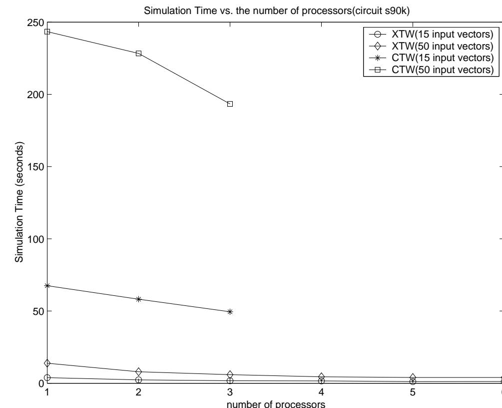

Fig. 10. simulation time vs. number of processors

4.1.2 Goodput and Committed Rate: Figure 11 depicts the good-put vs. the number of processors. Figure 12 shows the committed rate vs. the number of processors. Figure 11 shows that XTW has an almost linear increase in the good-put, while CTW has a relatively flat one. Figure 12 reveals the reason behind this phenomenon- XTW has a higher committed event rate than CTW. Moreover, XTW has an almost flat reduction in committed event rate when more processors are used, while CTW has a relatively steep reduction in its committed event rate. *These results indicate that XTW has a more efficient rollback mechanism.*

4.1.3 Speedup: Figure 13 shows *speedup vs. the number of processors*. It should be noted that the larger the throughput of a simulator, the harder it is to obtain a good speedup. Although XTW has a much larger goodput than CTW, the results indicate that XTW still has a bigger speedup than CTW

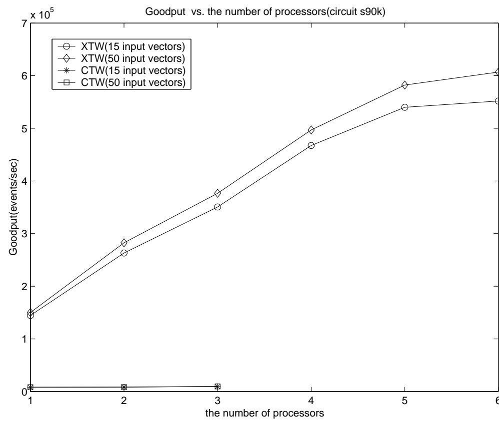

Fig. 11. goodput vs. the number of processors

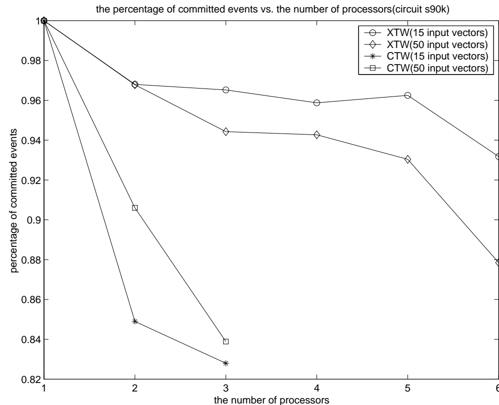

Fig. 12. committed events rate vs. the number of processors

in all the cases. Moreover, XTW has an almost linear increase in speedup while CTW has a relatively flat one. This clearly demonstrates that XTW has a smaller overhead than CTW.

4.2 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 [6], a Beowulf cluster with 128 Appro 1100i 1U nodes connected by a Myrinet. Each CLUMEQ node has dual Athlon 1900+ processors and 3G memory. Because CLUMEQ is a shared platform, the number of compute nodes which we used was limited to 40.

4.3 A Hierarchical Synthetic Benchmark Circuit Generation

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.

The benchmark circuits were generated as follows:

1) A modular-level netlist, consisting of real world circuits (which can be described in Verilog or VHDL) is created. This modular-level netlist is used to describe the connections among modules of the benchmark circuit.

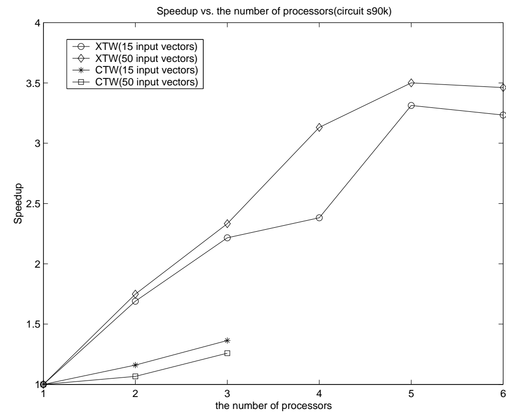

Fig. 13. speedup vs. the number of processors

2) Each node(module) of the modular-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.

4.4 A modular-level partition algorithm

The design of a large circuit follows a "divide and conquer" approach, in which the design is broken 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-modular 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 I shows the simulation time of three circuits simulated over 8 to 40 CLUMEQ nodes. The simulations of 25-million-gate over 12 and 8 nodes did not complete because of swapping. From the data in Table I, we see a consistent trend of a decrease in simulation time as the number of processors increases.

number of nodes	5-Million-gate	10-Million-gate	25-Million-gate
40	48.04	107.50	495.97
36	51.92	117.67	606.30
32	58.98	142.46	756.52
28	64.62	161.24	832.09
24	78.12	207.76	1058.69
20	106.36	261.49	1264.55
16	134.98	422.07	2187.50
12	200.59	1120.54	N/A
8	403.00	1324.84	N/A

TABLE I

SIMULATION TIME(SECONDS) VS. PROCESSOR NODES

Figure 14 shows the speedup vs. the number of processors. In Figure 14, we see that XTW scales almost linearly with

the number of processors and scales well with the size of the circuits.

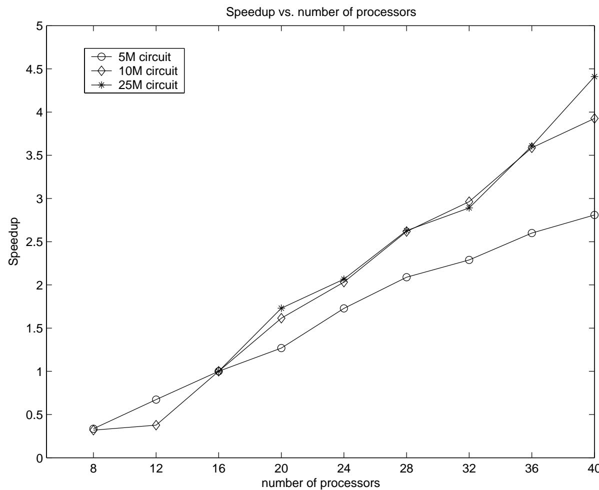

Fig. 14. Speedup vs. the number of processors

Figure 15 shows the Goodput vs. the number of processors. Goodput is the number of commited events per second. In Figure 15, we see that XTW's Goodput scales almost linearly with the number of processors. Nevertheless, Goodput decreases as the size of circuit increases. This due to the larger a circuit is, the more events are processed during a simulation. The increasing number of events adds more load on disk IO and network, thus, the throughput decreases.

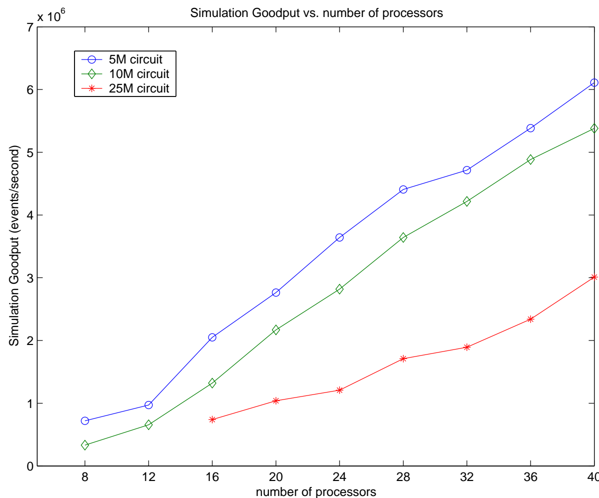

Fig. 15. Goodput vs. the number of processors

5 Conclusion

In this paper, two new mechanisms for improving the efficiency of distributed logic simulation were introduced. The first, *XEQ* is a 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 presume 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 were analyzed in theory. Comparisons to CTW revealed that XTW has a far superior performance. Experimental comparisons to a sequential version of XTW suggested that it is scalable, while experiments with large, synthetic circuits further support this claim.

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.

References

[1] D. Jefferson, "Virtual time," *Programming Languages and Systems*, 1985.

[2] H. Avril and C.Tropper, "On rolling back and checkpointing in time warp," *IEEE Trans. on Par. and Distr. Systems, vol.12, no.11, Nov.2001, pp. 1105-1122*, 2001.

[3] H.Avril and C.Tropper, "Scalable clustered time warp and logic simulation," *VLSI Design, Special Issue on Current Advances in Logic Simulation, Gordon-Breach, vol.19, no.3, lpp.291-313*, 1999.

[4] D.Martin and et al, "Analysis and simulation of mixed technology vlsi systems," *JPDC, Slpecial Issue Parallel and Distributed Discrete Event Simulation, pp. 468-493*, 2002.

[5] M. J. R.Chamberlain, "Parallel logic simulation of vlsi systems," *ACM Computing Surveys, vol.26, no.3, Sept. 1994, pp. 255-295*, 1994.

[6] www.clumeq.mcgill.ca, "Clumeq infrastructure," 2003.