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 comparisons to a sequential version of XTW reveal good speed-up.**

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 sending anti-messages.

VLSI 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 logic simulation in mind. As the name implies, in CTW LPs (representing gates) are gathered into clusters. Several techniques were developed for use with CTW 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 as well, and are an outgrowth of CTW.

A number of other efforts have been directed at VLSI simulation including [4] which employ optimistic algorithms and [5], which employ 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 TW. XTW was inspired by characteristics of discrete event circuit simulation. This new mechanism consists of a new event scheduling algorithm – *XEQ*– and a new rollback mechanism – *rb-message*. XEQ has O(1) cost bounded on the number of simulated entities (not on the number of events). *Rb-message* not only reduces the cost of annihilating previously sent messages, but also dramatically decreases the number of anti-messages – up to 13 times fewer rb-messages then anti-messages.

The remaining sections of the paper are organized as follows. Section2 describes the motivation for XTW. Section **??** 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

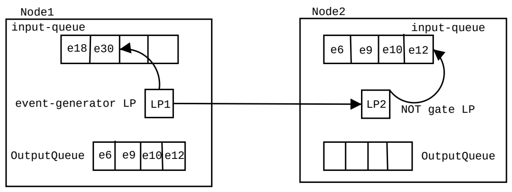

Fig. 1. a single LP, single InCh model in PDES

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.

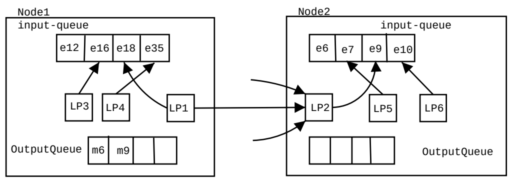

Fig. 2. a multiple LPs, multiple InChs model in PDES

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 [**?**]. Three techniques for checkpointing and rolling back in CTW are described in [?], 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 and its cost analysis.

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.

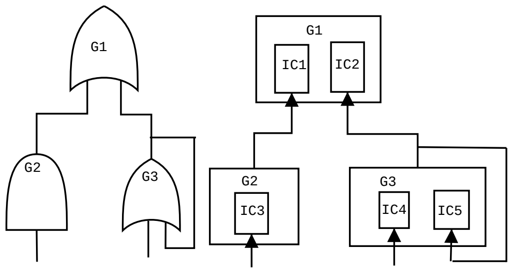

Fig. 3. Input Channel Model

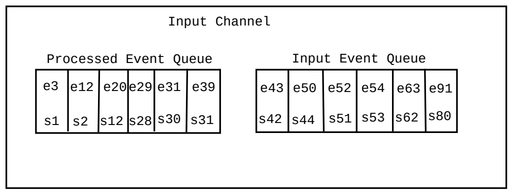

Fig. 4. The Structure of Input Channel

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 the virtual 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

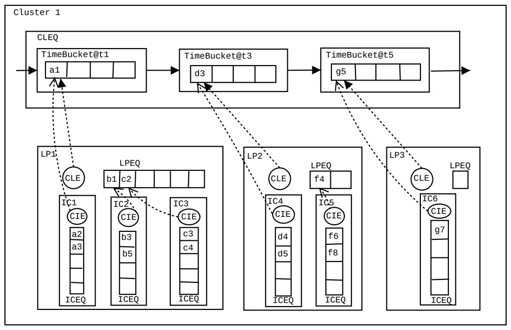

Fig. 5. The Structure of XEQ

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.

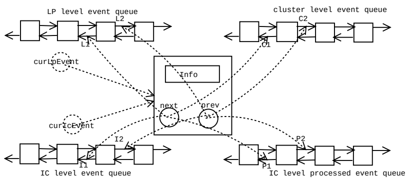

Fig. 6. an event node structure and its movement

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 :

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

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(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 order. We call the event has the smallest receive-time in an ICEQ or an ICPQ as the head event, and the event has the largest receive-time as the tail event. We define an *interted-point* as a position that a new event is inserted into, such that all the events with receive-time smaller than the new event is before the *interted-point* and all events with receive-time larger than the new event is after the *insertd-point*. If there is any event after the *interted-point*, the first event after the *interted-point* is called as the *after-inserted-point-event*.

In Time-Warp, an event causing rollback is called a straggler. In XTW, a new event can arrive at an *input-channel* in one of the six cases as depict in fig 7. *straggler* is an event which has a receive-time smaller than the Local Virtual Time(LVT) or has a send-time smaller than the send-time of the tail event of the ICPQ in its arriving *input-channel*. We do not distinguish between "messages" and "events" in the rest of the paper.

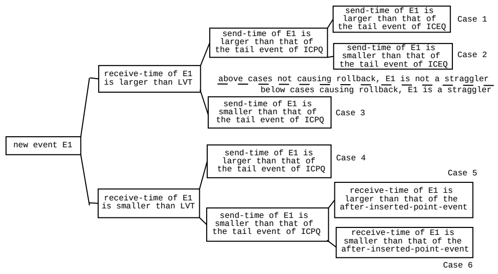
new event E1
receive-time of E1 is larger than LVT
receive-time of E1 is smaller than LVT
send-time of E1 is larger than that of the tail event of ICPQ
send-time of E1 is smaller than that of the tail event of ICPQ
send-time of E1 is larger than that of the tail event of ICEQ
send-time of E1 is smaller than that of the tail event of ICEQ
Case 1
Case 2
above cases not causing rollback, E1 is not a straggler
below cases causing rollback, E1 is a straggler
Case 3
send-time of E1 is larger than that of the tail event of ICPQ
Case 4
Case 5
receive-time of E1 is larger than that of the after-inserted-point-event
send-time of E1 is smaller than that of the tail event of ICPQ
receive-time of E1 is smaller than that of the after-inserted-point-event
Case 6

Fig. 7. *rb-messages* case analysis

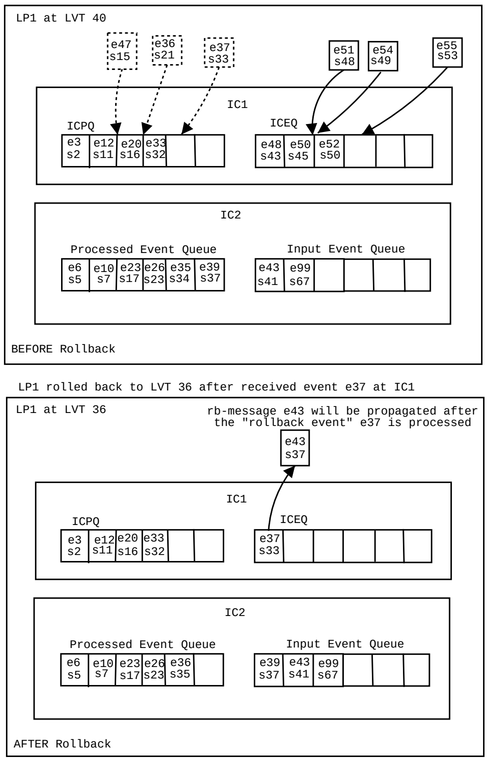

Fig. 8. *rb-messages*, an LP receives a straggler rb-message

Figure 8 depicts examples of the *rb-messages* mechanism. We describe how the *rb-messages* mechanism works in the XTW rollback procedure as follows:

After an event,Ev, arrives at an LP, it is handled in two cases as if it is a straggler or not.

If Ev is not a straggler, there are two cases:

1) Ev is not a straggler and its send-time is larger than the ICEQ tail event's send-time, e.g. e55 in figure 8. Ev then is scheduled in a normal way as described in section3.4.
2) Ev is not a straggler, but its send-time is smaller than the send-time of ICEQ's tail event, e.g. e54 in figure 8. Then Ev searches the ICEQ from tail to head to find the *"insert-point"* which is the position after the first event that has the send-time smaller than or equal to the Ev's. The *"insert-point"* could be at head of event. All events

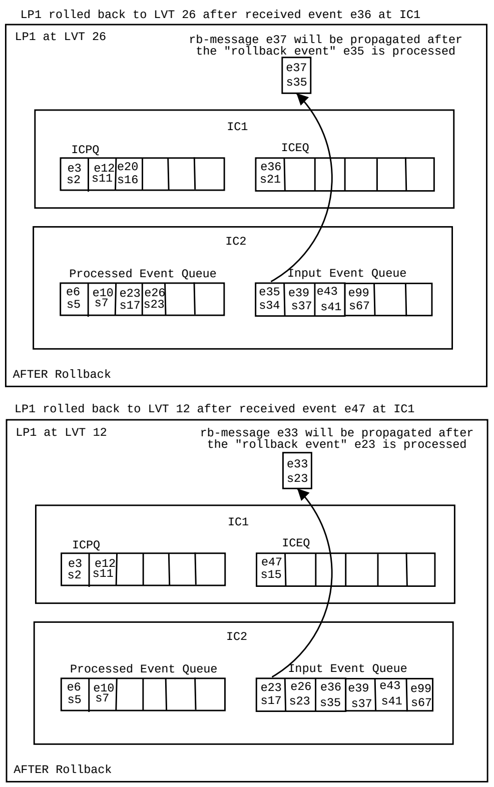

Fig. 9. *rb-messages*, an LP receives a no-straggler rb-message

that have send-time larger than Ev's are deleted from ICEQ. Ev then will be scheduled in a normal way as described in section3.4.

If Ev is a straggler, then XTW rollback procedure works in following steps:

1) The *current LP event* which is pointed by CLE is moved from CLEQ to the LPEQ.
2) The *current InCh events* pointed by respective CIE is moved from LPEQ to the head of respective ICEQs.
3) The *input-channel* which receives the straggler is rolled back and set the LP *rollback-time* accordingly as in following cases:
   a) Ev has the receive-time smaller than LVT and has the send-time larger than the send-time of the tail event in ICPQ, e.g., e37 at IC1 figure 8. Then the *input-channel*, which Ev arrives at, erases all events in its ICEQ. The LP *rollback-time* is set equal to the receive-time of Ev (See figure 8).
   b) Ev has the receive-time smaller than LVT and has the send-time smaller than the send-time of last event in ICPQ, e.g., e36 at IC1 in figure 8. Then Ev searches the ICPQ from tail to head to find the *"insert-point"*. We call the first event after the *"insert-point"* as Ei, such as, e33 at IC1. The LP *rollback-time* will be set in following two cases:
      i) The receive-time of Ei is larger than the receive-time of Ev, then the LP *rollback-time* is set equal to the receive-time of Ev. (See figure 9, LP1 at LVT 26)
      ii) The receive-time of Ei is smaller than the receive-time of Ev, then the LP *rollback-time* is set equal to the receive-time of Ei.

      In both cases, the *input-channel*, which Ev arrives at, erases all events in its ICEQ and all events after the *"insert-point"* in ICPQ. It should be noted that Ei is erased as well. (See figure 9, LP1 at LVT 26).
   c) Ev has the receive-time larger than LVT, but has the send-time smaller than the send-time of the tail event in ICPQ, e.g., e47 at IC1 in figure 8. Then Ev searches the ICPQ from tail to head to find the *"insert-point"*. The LP *rollback-time* is set equal to the receive-time of Ei. The *input-channel*, which Ev arrives, erases all events in its ICEQ and all events after the *"insert-point"* in ICPQ. (See figure 9., LP1 at LVT 12)
4) After the *rollback-time* is set, all *input-channels* which do not receive the straggler are rolled back by moving events which has receive-time smaller than or equal to the *rollback-time* from ICPQ to ICEQ.
5) The LP restores the states to the first state that has time-stamp smaller than or equal to the *rollback-time*.
6) Every *input-channel* submit one event to LPEQ if its ICEQ is not empty. LP submits one event to CLEQ. We 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. Rb-messages are normal output events and will be handled by subordinated LPs in the same way as described above.

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

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

Two sets of experiments are presented in this section:

- In subsection 4.2, a set of experiments compares CTW and XTW.
- In subsection 4.5, a set of experiments compares XTW and a sequential XTW simulator(XSS).

4.1 *Experimental Environment*

All experiments were conducted on a network of seven personal computers. Each computer is equipped with dual Pentium III 450 processors and 256 Megabytes of internal memory. The network is connected by a Myrinet switch which operates at one Gigabyte per second. XTW employs MPI as the software communication platform which guarantees a FIFO order in communication. All of the XTW experimental data presented is the average value from at least 100 runs, while each set of CTW data is the average value from at least 10 runs.

It should be noted that both XTW and CTW use one dedicated processor as a managing node which does data collection and GVT computing, however, none simulation computing is executed on this managing node. To make the experiment results clearer to illustrate the parallel trend, this managing node is not counted in the number of processors in all experiments result.

4.2 *XTW vs. Time Warp (CTW)*

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

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 benchmark circuit which consists of two s38584 and two 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. The average *Simulation Time* across the participating processors is presented.
- *Speedup*: *Speedup* is defined as the ratio of the simulation time of a simulator using one processors and the simulation time of the same simulator using more than one processors.
- *Throughput*: *Throughput* is defined as the number of processed events per second.
- *Good-put*: *Good-put* is defined as the number of committed processed events per second.
- *Committed Rate*: *Committed Rate* is defined as the ratio of the *Good-put* 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.

4.2.1 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.

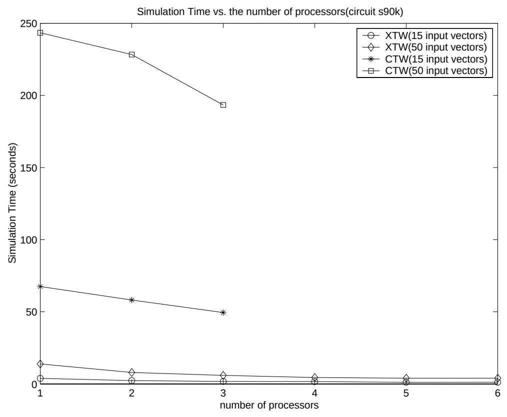

Fig. 10. simulation time vs. number of processors

4.3 *Good-put 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.

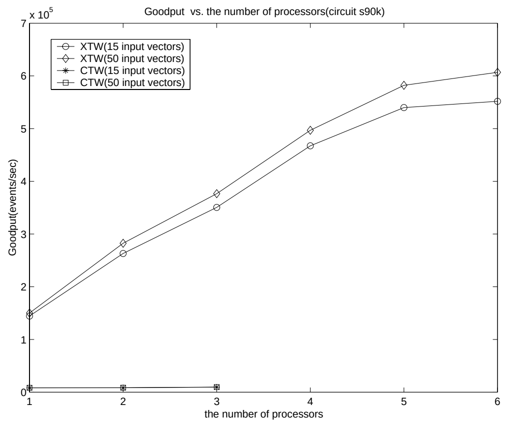

Fig. 11. good-put vs. the number of processors

4.4 *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 in all the cases. Moreover,

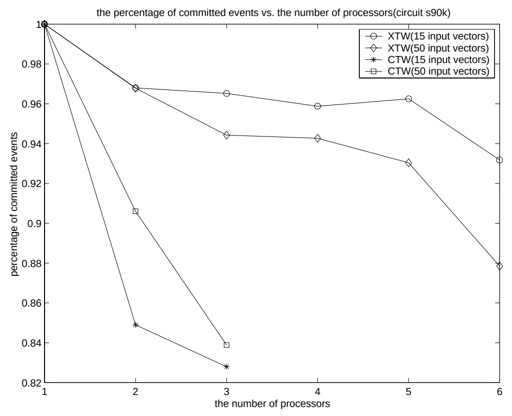

Fig. 12. committed events rate vs. the number of processors

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.

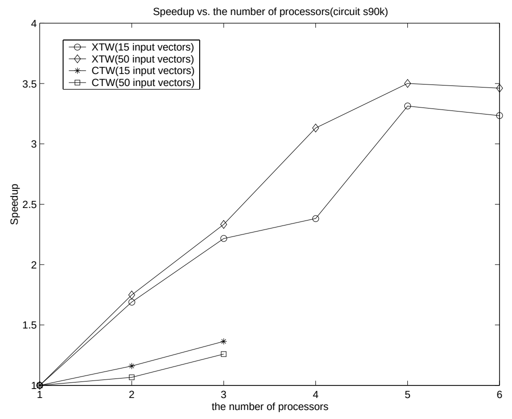

Fig. 13. speedup vs. the number of processors

4.5 *XTW vs. Sequential Simulator*

***********You need to change NTW to CTW in figures 10 and 11. Also, how does the maximum simulation time across all of the processors differ from the simulation time? If there is a difference, why are you using the maximum simulation time?****************

In this subsection, several benchmark circuits are simulated by both XTW and a sequential simulator. The purpose of these experiments is to compare the performance between parallel and sequential simulations.

4.5.1 The Sequential Simulator: The sequential simulator actually is a sequential version of XTW which implements the same event-scheduling and logic simulation algorithms as does parallel XTW. The sequential simulations are processed on one of the cluster PCs with a single processor.

4.5.2 Benchmark Circuits and Metrics: Three benchmark circuits were used in the experiments. They are as follows:

- s38584 circuit with a total of 20996 gates
- s180k consisted of four s38584 and four s38417 circuits with a total around 180,000 gates
- s360k consisted of eight s38584 and eight s38417 circuits with a total around 360,000 gates

The performance metrics which we employ are defined as follows:

- *max simulation time* is defined as the maximum elapsed real time across the participating processors for each simulation. The partitioning time is included in *max simulation time*.
- *peak memory usage* is defined as the maximum peak memory usage across the participating processors for each simulation.
- *absolute speedup* is defined as the ratio of the sequential *simulation time* to the *max simulation time* for a parallel execution.

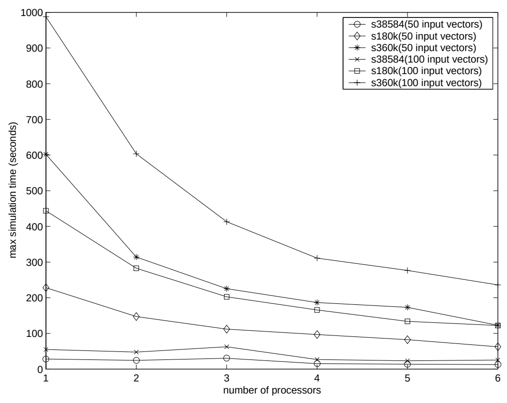

Fig. 14. max simulation time vs. number of processors

4.5.3 Max Simulation Time, Absolute Speedup and Good-put: Figure 14 shows the *max simulation time* vs. the number of processors. The one processor point is the result of the sequential simulator. In Figure 14, we can clearly see a trend that the *max simulation time* decreases as the number of processors increases. Moreover, this trend is enhanced as the size of circuit and the number of vectors are increased (e.g. simulations for s180k and s360k). However, the *max simulation time* of simulations for s38584, which is a relatively small circuit, only decreases slightly as the number of processors increases and has a bump at the point of 3 processors due to the unbalanced load across processors. It should be noted that the *max simulation time* of simulations for the s360k circuit decreases steeply from one processor to two and more processors due to the swap-memory used in the sequential simulations.

Figure 15 and Figure 16 present respectively the *absolute speedup* and the *good-put* vs. the number of processors for three benchmark circuits simulated with 50 vectors and 100 vectors. In both figures, there is a general trend of increasing speedups and good-puts with an increasing number of processors, circuit size and number of vectors. A slight drop of *absolute speedup* in s38584 with 3 processors is due to the unbalanced loads assigned across the processors.

The trends in Figure 14, Figure 15 and Figure 16 all clearly indicate that XTW is scalable and is capable of simulating

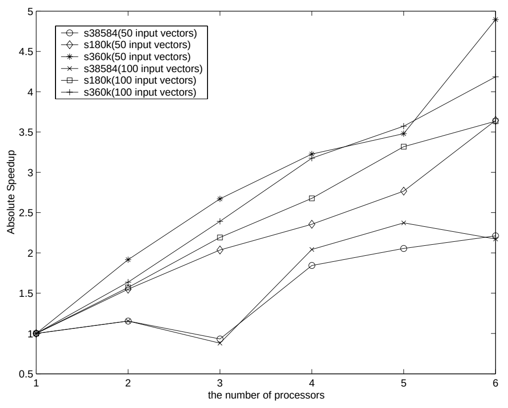

Fig. 15. absolute speedup vs. number of processors

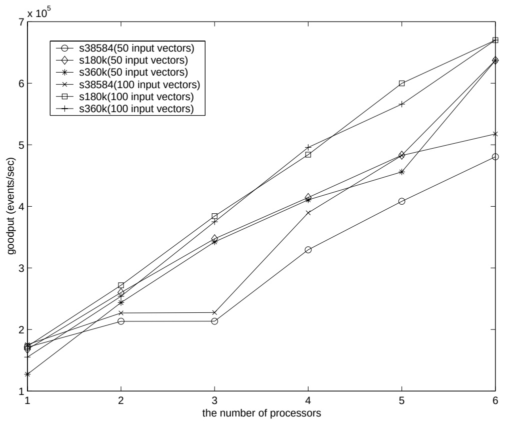

Fig. 16. good-put vs. number of processors

large circuits.

4.5.4 Peak Memory Usage: To quantify parallel *peak memory usage*, we consider the following metric: *Peak memory usage ratio(PMUR)* is defined as the ratio of the *peak memory usage* of a parallel simulation to that of a sequential simulation. Let PPMU be the parallel peak memory usage and SPMU be the sequential peak memory usage.

Peak Memory Usage Ration = Parallel Peak Memory Usage/Sequential Peak Memory Usage

Figure 17 displays the results of the *peak memory usage ratio* for the two large circuits –s180k and s360k. In Figure 17, although there are some increases due to unbalanced loads across the processors, we can clearly see a general trend of decreasing peak memory usage ratios with an increase in the number of processors for all simulations. This trend indicates that *XTW* is capable of simulating large circuits that the sequential simulator is not capable of simulating because of insufficient memory in a single machine.

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 results in an $O(1)$ event scheduling. The second, *rb-messages*, reduces the

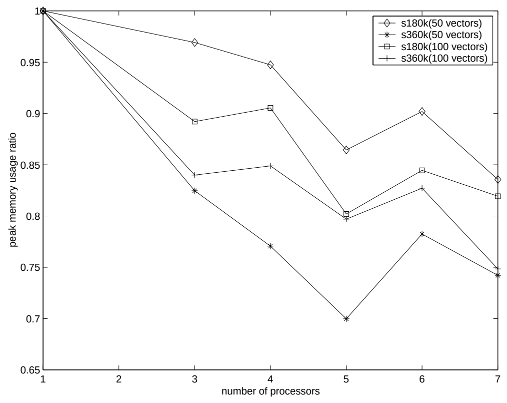

Fig. 17. peak memory usage ratio vs. the number of processors(s180k and s360k)

rollback and message cancellation 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, *XTW*.

The cost of these algorithms were analyzed in theory. Comparisons to CTW revealed that XTW has a far superior performance. Comparisons to a sequential version of XTW revealed its scalability.

It is certainly desirable to make use of XTW on larger circuits and to modify it for use in behavioral and mixed behavioral/logic simulation. 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.