This is to certify that the

dissertation entitled

LOGIC SIMULATION
ON MASSIVELY PARALLEL SIMD MACHINES

presented by

Yunmo Chung

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

Major professor

Date 2/10/92

DATE DUE	DATE DUE	DATE DUE

LOGIC SIMULATION ON MASSIVELY PARALLEL **SIMD** MACHINES

By

Yunmo Chung

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1991

Sim
VLSI d
driven s
In this t
efficient
parallel
the propo
1 (MasPa
queue siz
performan
gate. The
increase as
logic simul
conservativ
techniques
the new tec
token driven
simulation w
thesis presen
environments
simulation, as

ABSTRACT

LOGIC SIMULATION ON MASSIVELY PARALLEL SIMD MACHINES

By

Yunmo Chung

Simulation is the primary tool for validation and analysis of digital circuits in VLSI design. As the complexity of VLSI circuits has increased, distributed event-driven simulation has attracted considerable interest for providing fast simulation. In this thesis, based on the characteristics of gate-level logic simulation, we study efficient paradigms and data structures for fast parallel logic simulation on massively parallel SIMD (Single Instruction Multiple Data) machines. The performance of the proposed schemes is measured on the CM-2 (Connection Machine) and the MP-1 (MasPar) in terms of number of simulation cycles, parallelism, maximum event queue sizes, and execution times. We present a probabilistic model to estimate the performance of parallel logic simulation when a processor contains more than one gate. The performance estimation shows that parallelism and processor utilization increase as the number of gates per processor increases. We propose new parallel logic simulation techniques by giving a clock advancement window to each gate in conservative simulation and optimistic simulation. Experimental results show that the techniques give much better performance than traditional simulation techniques since the new techniques enhance simulation clock advancements. We present a distributed token driven logic simulation technique as a parallelized version of compiled-code logic simulation which is used to verify the functional correctness very efficiently. This thesis presents broad studies of parallel logic simulation in massively parallel SIMD environments. The research results are useful in the implementation of parallel logic simulation, as a part of VLSI design, in SIMD processing environments.

To my parents and my wife

I w

his con

suppor

I w

Dr. Ri

guidan

My

during

especia

my rese

daughte

days. I

their pe

I wo

Laborat

Illinois a

ACKNOWLEDGMENTS

I wish to express my appreciation to my thesis advisor Dr. Moon Jung Chung for his consistent guidance and encouragement right from the beginning, and his financial support. I am grateful for the many discussions and invaluable comments he provided.

I would like to thank my guidance committee members, Dr. Anthony S. Wojcik, Dr. Richard Enbody, and Dr. Joseph C. Gardiner, for their help, encouragement and guidance.

My sincerest thanks goes to the Korean government for their financial support during the beginning of my Ph.D. program. I thank friends in our department, especially Mr. J. Engelsma, Mr. R. Baldwin, and Mr. P. Wolberg, for helping me do my research and write my dissertation. I thank my wife Yunwha and my two pretty daughters, Youjin and Yuri, for their support, patience, and love during many long days. I express my special thanks to my father-in-law and mother-in-law in Korea for their persistent spiritual encouragement and help.

I would like to acknowledge Thinking Machines Corporation, Argonne National Laboratory, and the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, for allowing the use of their resources.

List of

List of

1 Intr

1.1

1.2

1.3

1.4

1.5

1.6

2 Paral

2.1 P

2.2 B

2.

Table of Contents

2.3

2.4

3 Perf

3.1

3.2

3.3

3.4

4 Perfo

4.1

4.2

4.3

4.4

List of Figures

Cha

Intr

Simulati
circuit si
difficult
number
circuit s
needed
Thus, ga
time an

Trad
fication
tion usi
In an a
tracted
the sub

As
special-
develop
simulat

As
process
has bee
Instruc
SIMD
simula

Chapter 1

Introduction

Simulation is the primary tool for validation in VLSI design. Though low-level analog circuit simulation is used, such circuit simulation has several drawbacks [36]. It is difficult to simulate the entire system for most digital integrated circuits with a large number of logic gates because extensive computer time is required. Also, analog circuit simulation generates traces of voltage levels, which are more detailed than needed to validate digital designs. For digital designs, binary signals are sufficient. Thus, gate-level *logic simulation* has been developed, as an effort to reduce simulation time and provide efficient validation of digital circuits.

Traditional event-driven simulation techniques have been widely used for the verification of timing and functional correctness of circuit designs. However, logic simulation using these techniques becomes slow as the complexity of VLSI circuits increases. In an attempt to cope with the problem, parallel logic simulation has recently attracted a considerable amount of interest. Simulation time can be reduced based on the substantial amount of parallelism achievable.

As one approach for fast logic simulation, hardware logic simulation, which uses special-purpose hardware that executes many simulation steps in parallel, has been developed to speed up the simulation process [2, 45]. But, since hardware logic simulation is very expensive, it is not applicable as a common tool for simulation.

As the computer technology has been developed for parallel processing on multiprocessor environments, research on parallel simulation techniques for fast simulation has been active. So far, most of that research has been related to MIMD (Multiple Instructions Multiple Data) processing environments. Recently, massively parallel SIMD machines have become available. In this thesis, characteristics of both logic simulation and massively parallel SIMD machines are investigated to develop efficient

paradigms
lation is s
small-grai
Par) and
machines

1.1

In VLSI
particular
correctne
siderable
can be cl

- Cir
  of
- Sw
  for
- Sw
  or
  ma
- Ga
  ra
  O
  ze
- Fu
  of

paradigms and data structures for fast parallel logic simulation. Gate-level logic simulation is suitable for massively parallel SIMD machines since logic simulation requires small-grain operations and may need many processors. Currently, the MP-1 (MasPar) and the CM-2 (Connection Machine) are available as massively parallel SIMD machines with enough local memory for logic simulation.

1.1 Computer Simulation in VLSI Design

In VLSI design, simulation results are produced for a given circuit design with a particular set of input vectors. Then, these results are used to verify the functional correctness and timing of the circuit design. To achieve reliable circuit design, considerable time is often invested in computer simulation. Types of digital simulation can be classified as follows [14, 36, 72]:

- Circuit simulation : This simulation is used for accurate simulation in the design of analog circuits, such as filters, comparators, and operational amplifiers.
- Switch-level simulation : Transistors with constant delay are used as primitives for simulation.
- Switch-level with timing : To increase the chances of catching race conditions or to compute propagation delays in switch-level simulation, some timing information is used.
- Gate-level logic simulation : Gate-level simulation is faster and nearly as accurate as switch-level with timing. A small set of functions, such as NOT, AND, OR, NAND, and NOR gates are supported. Gate-level simulation supports zero-delay, unit-delay, or fixed-delay.
- Functional and register-transfer simulation : A designer describes the behavior of components of a circuit design at a high level. Simulation is performed based

on t

tion

such

- Mix

gate

of a

spee

- Mix

ana

1.2

This thes

There are

informati

The f

logic sim

on delay

descripti

language

Each pro

indepenc

increases

of execu

manager

function

The

logic sim

on these high-level components. The high-level functions indicate transformations on data as the data moves from one storage device or register to another such device.

- Mixed-level simulation : Different levels of simulation—switch, switch-timing, gate, and function—are allowed to be run simultaneously. This allows portions of a design to be tested in detail, while other portions are tested with greater speed but less detail.

- Mixed-mode simulation : Simulation is simultaneously performed using both analog and digital modes.

1.2 Gate-Level Logic Simulation

This thesis investigates how to use parallel simulation paradigms for logic simulation. There are two kinds of logic simulation techniques classified according to how timing information is handled.

The first approach, compiled-code logic simulation, is used for fast and simple logic simulation where the delay of logic gates is not critical, i.e. simulation based on delay is not required. Zero-delay or unit-delay logic simulation is performed. The description of a circuit to be simulated is translated into a conventional computer language, such as assembly or Pascal code. The code is compiled and then run [59]. Each process is evaluated once for every input vector during each simulation cycle independent of the input patterns. Compiled-code simulation provides significant increases in performance over event-driven simulation by means of the high speed of execution of compiled-code. Since this technique does not require event queue management, it is extremely efficient. Compiled-code simulation can be used for functional simulation of combinational and synchronous circuits only.

The other simulation technique, event-driven simulation, is used for multi-delay logic simulation. So far, event-driven simulation has been used to implement several

types of sim

tion is effici

the selectiv

timing anal

1.2.1 C

Logic simul

queuing ne

be develop

Gate D

We can

value of th

VHDL [1]

port delay

the delay

acteristic

switching

either the

pulse dura

only trans

Non-p

During

the proce

us to ass

schemes [

Fixed

In log

determin

types of simulators. Processes communicate by sending events. Event-driven simulation is efficient since it works by evaluating only the active elements, which is called the selective trace approach. An additional advantage is that we can easily deal with timing analysis and asynchronous designs.

1.2.1 Characteristics of Logic Simulation

Logic simulation has special characteristics which are not usually displayed in general queuing network simulation. In this thesis, event scheduling and data structures will be developed which utilize these properties.

Gate Delay

We can assume that the delay of each gate is independent of the input and output value of the gate. In other words, each gate has a fixed delay during simulation. VHDL [1] recognizes two common types of gate delay: *transport* and *inertial.* Transport delay is a characteristic of hardware devices whereby a pulse is transmitted after the delay no matter how long the duration of the pulse is. Inertial delay is a characteristic of switching circuits whereby a pulse whose duration is shorter than the switching time of the circuit will not be transmitted at all. Most logic simulators use either the unit delay or the transport delay model [24, 62, 71]. In our simulation, pulse durations are assumed to be no shorter than the switching time. Therefore, only transport delay is considered.

Non-preemption

During the execution of a process on a processor, there is no interruption of the process in execution before its termination. The transport delay model allows us to assume that there is no preemption of output events, like other simulation schemes [25, 32, 46].

Fixed Propagation Routing

In logic simulation, each active gate propagates the same events to *all* of its predetermined successors. This property simplifies the data structures in distributed

event-dr
tion for

Look

Looka
happen i
is allowe
tion appl
mance. C
to achiev
capability

Fine

In logi
ality and
etc., is do
nique app
managem
operations

Large

A circu
tion, a gat
of a large
become m
sively para
not be seri

event-driven logic simulation protocols since we do not need to consider any condition for propagation directions to successors.

Lookahead Capability

Lookahead refers to the ability to predict what will happen and what will not happen in the simulated future [32, 33]. Due to the fixed delay of each gate, a gate is allowed to predict the output event time when it receives an event. If a simulation application has good lookahead capabilities, we can often obtain better performance. Conservative simulation relies on lookahead more than optimistic simulation to achieve good performance [34]. A gate in logic simulation has a good lookahead capability since the gate has a predefined delay.

Fine Grain Operations

In logic simulation, each gate performs event evaluation according to its functionality and event propagation. Event evaluation for logic gates, such as AND, OR, etc., is done by a simple table lookup operation. Depending on the simulation technique applied, some additional operations, such as queue manipulation, and rollback management, are required. Event evaluation operations take less time than other operations in a simulation cycle.

Large Number of Elements

A circuit to be simulated consists of gates and links. In event-driven logic simulation, a gate is considered a process. When a coarse grain machine is used, simulation of a large circuit leads to partitioning and load balancing problems. The problems become more serious as the complexity of VLSI circuits increases. If we use a massively parallel SIMD machine with a large number of processors, the problems may not be serious.

1.3

In thi
simul

1.3.1

There
asynch
clock.
queue
smalle
accord
into tw

Synch

In syn
events
timest
The ev
execut
62]. H
limitin
replace

Conse

The C
paradig
to the
metho
propose

1.3 Parallel Simulation Paradigms

In this section, distributed simulation paradigms which can be used for parallel logic simulation are discussed.

1.3.1 Event-Driven Simulation

There are two approaches for distributed event-driven simulation: *synchronous* and *asynchronous*. In synchronous simulation, all processes are synchronized by a global clock. In asynchronous parallel simulation, each process maintains its own message queues and local simulation clock. The local simulation clock of a process is the smallest timestamp of events to be processed. A process executes arriving events according to a certain scheduling policy. Asynchronous simulation can be divided into two common simulation paradigms: *conservative* and *optimistic*.

Synchronous Simulation

In synchronous simulation, there is a central event queue, which contains all of the events during simulation. A *Global Virtual Time* (GVT) is computed as the smallest timestamp of all unprocessed events. All processes are synchronized by the GVT. The events with timestamps equal to the GVT are chosen from the event queue, and executed in parallel. Similar approaches have been explored for logic simulation [55, 62]. However, the centralized nature of the queue causes a communication bottleneck limiting the performance. For efficient parallel logic simulation, the central queue is replaced by distributed event queues.

Conservative Simulation

The Chandy-Misra approach [20, 21] is the most well-known conservative simulation paradigm. In the Chandy-Misra approach, each process executes events according to the input waiting rule so that earlier events are guaranteed not to occur. The method has an inherent deadlock management problem. Two strategies [20] have been proposed to handle the deadlock problem: *deadlock avoidance protocol* and *deadlock*

detection and recovery paradigm. In the deadlock avoidance protocol, null messages are used to propagate clock information to avoid deadlock. In the deadlock detection and recovery paradigm, simulation is allowed to deadlock, the deadlock is detected, and then the deadlock is broken. Several other conservative simulation approaches have been developed in [6, 22, 30, 54].

Optimistic Simulation

In Time Warp [42, 43], the most popular optimistic paradigm, a process may execute each event as soon as the event arrives. A local virtual time (LVT) at a process is defined as the smallest timestamp of unprocessed events at the process. Whenever a process receives an event with a timestamp less than the LVT of the process, it rolls back immediately to the state just before the timestamp of the newly received event. GVT acts as the floor on all future rollbacks. To handle the rollback, previous states must be saved. Two ways of controlling the cancellation of side effects given are discussed in [43]. They are lazy cancellation, which cancels side effects in the next forward simulation, and aggressive cancellation, which cancels side effects as soon as they are found. *Fossil collection* is concerned with recovering storage associated with past simulation times by discarding processed events with timestamps less than or equal to GVT. A variation of Time Warp, which runs in a synchronous parallel environment, has been proposed in [64].

Moving Time Window

Moving Time Window (MTW) has been proposed in [61] to reduce rollback frequency and the space overhead of the Time Warp mechanism. The MTW technique uses a global window to limit the maximum difference between the local virtual simulation time of communicating processes. In other words, the global time window is applied to allow only those simulation events whose execution times fall within the window to be considered for execution.

1.3.2

In dema
demand
in distri
the over
particul

1.3.3

Token-d
no timi
of event
only ca
may be
simulat
schedul
least or
signal v
of main

1.4

Paralle
in seve
asynch
of MIM
paralle
techno
MIMD
grain o

1.3.2 Demand-Driven Simulation

In demand-driven simulation [60], gates are only evaluated when their outputs are demanded. This eliminates many of the unnecessary events which are generated in distributed event-driven simulation. The main disadvantage of this approach is the overhead incurred by a recursive backtracking routine. The recursion becomes particularly inefficient when the circuit is deep.

1.3.3 Token-Driven Simulation

Token-driven simulation [28] can be used to verify functional behavior, but provides no timing information. In token-driven simulation, tokens are propagated instead of events. Unlike events, which are time-stamped with a simulation time, a token only carries the output signal from a process without a timestamp. An identifier may be assigned to each token, if necessary, for synchronizing messages for correct simulation. Instead of evaluating processes based on the occurrence of a previous scheduled timestamped event, process evaluation is triggered by the presence of at least one token in each of the gate's input queues. A token may contain one or more signal values. By eliminating timestamps, we also eliminate the non-trivial problem of maintaining global time over multiple processes.

1.4 Massively Parallel SIMD Machines

Parallel simulations on MIMD machines have some advantages over SIMD machines in several aspects: general computational capabilities; larger-grain parallelism and asynchronous computation on each processor. However, one significant drawback of MIMD machines is that they usually have much fewer processors than massively parallel SIMD machines. SIMD architectures are the only alternative, with present technology, to achieve massive parallelism. They offer significant advantages over MIMD architectures for parallel logic simulation. Logic simulation requires small-grain operations, such as NOT, AND, and OR. A large circuit requires a large number

of proce
architec
of logic
balancin
fewer ga
instructi
and data

1.4.1

There a
Processo
CM-2 a
has eno
two pro
using th
16K pro

Local

The CM
of the lo
machine
each no
operatir
simulati
required

Relate

Some cr

of processors, which is well suited to a massively parallel SIMD environment. SIMD architectures also offer simpler synchronization, which is crucial to the performance of logic simulation. A SIMD environment has no serious circuit partitioning or load balancing problems since, in general, each processor of a SIMD machine will simulate fewer gates than for a MIMD machine. In a SIMD architecture, however, different instructions cannot be issued at the same time, which limits simulation techniques and data structures.

1.4.1 Considerations on Massively Parallel SIMD Machines

There are several massively parallel SIMD machines, such as the DAP (Data Array Processor), the MPP (Massively Parallel Processor), the CM-2, and the MP-1 [5]. The CM-2 and the MP-1 are used in this thesis as target machines since each processor has enough local memory for event queues and direct communication between any two processors is possible. The majority of our experimental results were obtained using the CM-2. The CM-2 [66] and the MP-1 [11] consist of 4K to 64K and 8K to 16K processors, respectively.

Local Memory

The CM-2 and the MP-1 have 64K and 128K bits of local memory, respectively. Most of the local memory capacity on each physical processor of a massively parallel SIMD machine is available for user data. In SIMD, no program resides in the nodes while each node in MIMD needs memory space to store some system software, such as an operating system. But the size of each local memory in SIMD is relatively small for simulation of a large number of input vectors. Thus, efficient data structures are required to maximize available queue size.

Related Instructions

Some critical instructions related to logic simulation are explained below.

- direct *send*

  This instruction sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. As one of the most common and time-consuming instructions, this instruction is used to propagate a generated event from a node to its successor nodes. Many processors sending to the same place results in *conflicts*.

- *global-min*

  This instruction, called *reduction*, is for computation or communication between a front-end processor and local processors. One value is examined in every selected processor, and the smallest of all these values is returned to the front end. This instruction is used to compute the global clock by obtaining the minimum local virtual time of all active gates.

- *array access*

  The CM-2 supports two array access instructions, *aref32* and *aset32*, to read and write an element of an array in each processor respectively. Since each processor of the CM-2 is bit-serial, these take a relatively long time compared with the read/write operations of non-array variables. They are frequently used for the manipulation of event queues.

The performance of the instructions on the CM-2 and the MP-1 will be given in Chapter 3.

1.5 Thesis Motivation

Today, simulation constitutes at least half of the design cycle. As the VLSI designs become larger, the simulation cost can only grow. Of course, the amount of validation required increases due to more interconnections between elements.

The p
structures
and massi
massively
Therefore
is studied
Curren
tations. C
machines
To cope v
gates to t
performan
We in
ficient eve
simulation
a window
window a
are sent a
size on pe
cle (or ite
propagati
Thoug
simulation
of digital
behavior i
code simu
tokens to

The primary purpose of this thesis is to propose efficient paradigms and data structures for parallel logic simulation based on the characteristics of logic simulation and massively parallel SIMD machines. For example, an event queue manipulation on massively parallel SIMD machines is critical for the performance of logic simulation. Therefore, an efficient event queue structure for each distributed simulation protocol is studied.

Currently, each processor can simulate only one gate due to local memory limitations. One of the critiques of parallel logic simulation on massively parallel SIMD machines is that parallelism is low compared with the number of processors used. To cope with this problem, we investigate the effect of the ratio of the number of gates to the number of processors. We develop a probabilistic model to estimate the performance of parallel logic simulation when we increase this ratio.

We investigate the enhancement of local simulation clock advancements for efficient event evaluation and propagation in conservative simulation and optimistic simulation. We present advancement window concepts, in which each gate computes a window containing events to be involved in event evaluation. All events within the window are evaluated during a simulation cycle, and the resulting sequence of events are sent as a single message for fast simulation. We investigate the effects of window size on performance. More than one event evaluation is allowed for a simulation cycle (or iteration) to enhance local clock advancements. In addition, more than one propagation event can be sent in a message to save communication cost.

Though compiled-code simulation has been proposed for fast zero (or unit)-delay simulation, such a simulation works only on sequential machines. As the complexity of digital circuits has increased, faster logic simulation for the verification of circuit behavior is required. We investigate the possibility of a parallel version of compiled-code simulation. As one approach, we propose a parallel simulation technique using tokens to synchronize all the signals from the same input vectors.

1.6

The rem
ficient si
on distri
machine
ulation.
simulati

In Cl
Chapter
lation cy
of the te
CM-2 ar

In Cl
can beco
model is
on the g
of simul
with exp
simulati

In C
performa
window
possible.
is given.
in terms
In additi
how a se

In Ch
driven log

1.6 Thesis Overview

The remainder of this thesis is organized as follows. In Chapter 2, we present efficient simulation paradigms and data structures for parallel logic simulation based on distributed discrete event-driven simulation protocols on massively parallel SIMD machines. The event-driven simulation protocols investigated are synchronous simulation, conservative simulation, and optimistic simulation. Some variations of the simulation protocols are proposed to improve performance.

In Chapter 3, the performance of parallel logic simulation paradigms proposed in Chapter 2 will be measured on the CM-2 and the MP-1, in terms of number of simulation cycles, parallelism, maximum queue sizes, and execution times. Characteristics of the test circuits will be discussed. In addition, a performance comparison of the CM-2 and MP-1 is presented.

In Chapter 4, we prove that the parallelism on massively parallel SIMD machines can become high if we increase the gate-to-processor ratio. In addition, a probabilistic model is proposed to estimate the performance of parallel logic simulation based on the gate-to-processor ratio. The model is used to predict parallelism, number of simulation cycles, and execution times. The predicted performance is compared with experimental results for three distributed simulation protocols, i.e, synchronous simulation, conservative simulation, and optimistic simulation.

In Chapter 5, we present techniques to enhance clock advancement for better performance in conservative simulation and optimistic simulation. An advancement window at each gate is computed to allow more than one local clock advancement if possible. A method to compute and use windows in the above simulation protocols is given. The performance, as a function of window size, is measured on the CM-2, in terms of number of simulation cycles, parallelism, queue size, and execution times. In addition, communication costs according to message lengths are measured to see how a sequence of events sent as a single message can save communication costs.

In Chapter 6, we present a new parallel logic simulation protocol called token-driven logic simulation, which is efficient for zero-delay or unit-delay logic simulation.

Many pe
simulatic
simulatic
driven si
We show
mental re
SIMD ma
problems

Many performance considerations are discussed. The performance of token-driven simulation is measured on the CM-2. We compare the performance of token-driven simulation and compiled-code simulation. In addition, the performance of token-driven simulation on the BBN TC-2000, a shared-memory MIMD machine, is given. We show that, with a reasonable partitioning scheme, the simulation gives experimental results comparable to those obtained by a similar algorithm running on an SIMD machine. Finally, we conclude our thesis and suggest some interesting research problems which remain.

Chapter 2

Parallel Logic Simulation in SIMD Environments

This chapter presents efficient simulation paradigms and data structures for parallel logic simulation based on the characteristics of gate level logic simulation. The event-driven simulation protocols investigated are synchronous simulation, conservative simulation, and optimistic simulation. Some variations are considered to implement the simulation techniques efficiently on massively parallel SIMD machines.

Efficient event queue manipulations are critical on massively parallel SIMD machines since array access operations are, in general, slow. Circular FIFO (First-In-First-Out) lists are used as event queues in both synchronous and conservative simulation. As an efficient event queue structure for optimistic simulation, we present a circular binary search queue structure which allows binary search on a circular list. Advantages and disadvantages of the proposed techniques will be discussed.

In optimistic simulation, both lazy and aggressive cancellation techniques work poorly in SIMD processing environments. Thus, a new cancellation scheme, called immediate cancellation, is also presented.

The remainder of this chapter is organized as follows. Section 2.1 discusses the literature survey of distributed event-driven simulation, with special emphasis on parallel logic simulation. Basic data structures for designing efficient parallel simulation paradigms on massively parallel SIMD machines will be discussed in Section 2.2. Section 2.3 presents efficient simulation paradigms for logic simulation.

2.1 Prior Work

A simple approach to parallel implementation of logic simulation is to use vector machines. However, it has been shown in [18, 58] that vectorization techniques for logic simulation can achieve very limited parallelism. Distributed event-driven simulation techniques offer the most promise.

In the well-known Chandy-Misra conservative simulation algorithm, deadlock management is a major problem in MIMD environments, and has been investigated in [30, 32, 58, 63]. Soule and Gupta [63] classified the types of deadlocks in digital circuit simulation to reduce deadlock occurrence, and reported simulation results on a Multimax. They observed that conservative simulation with null messages is inefficient on the MIMD machine. In [74], a new conservative simulation algorithm, called YADDES, is proposed, which uses a dataflow network to avoid deadlocks.

In Time Warp, several techniques have been proposed in [47, 50] to reduce the storage overhead and rollback problems. Moving Time Window (MTW) was proposed in [61] as a semi-optimistic approach to reduce the rollback frequency and the space overhead of Time Warp.

Comparisons of the performance of synchronous simulation, conservative simulation, and optimistic simulation techniques have been reported based on actual implementation on MIMD machines: Transputer-based multiprocessor with 8 nodes [57] and BBN Butterfly with 64 processors [34]. Fujimoto [34] reported that Time Warp outperforms the Chandy-Misra algorithm in his implementation. Lin *et al.*, proposed a performance comparison model of parallel logic simulation in a MIMD environment [48]. They showed that Time Warp always outperforms the Chandy-Misra algorithms under the assumptions of zero overhead of rollback and state saving. It has been known that simulations using Time Warp on MIMD machines give promising performance in battlefield simulation [37], digital hardware simulation [8], and producer/consumer simulation workloads [4].

When optimistic simulation is implemented on MIMD machines, fast computation of Global Virtual Time (GVT) is very important, but it is not easy. Preiss suggests

in [56] a token-ring calculation of GVT. Briner presents in [14] a GVT approximation algorithm using a tree of processors. Baldwin *et al.* present a GVT computation algorithm with overlapping window concept in [7].

A few results are available for parallel logic simulation in SIMD environments. Results have been published on parallel simulation of queuing models [49], circuit simulation [71], and switch-level simulation on the Connection Machine [15]. We have investigated logic simulation schemes on a massively parallel SIMD machine. Since queue structures for optimistic simulation on massively parallel SIMD machines are so important, we have developed several schemes as event queue structures. For example, a data parallel queue scheme was presented in [24]. In this scheme, an event is assigned to an event processor and event evaluation is done based on a data parallel approach. Next, a single queue scheme was presented which assigns an event queue to a gate in [27, 29]. In the implementation, a variation of Time Warp was presented which uses a single queue at each process and computes a lower bound of rollback based on immediate cancellation to reduce space overhead [29]. The analysis of local clock advancement in a SIMD environment, and the effect of moving time window size on execution time were also reported in [27]. In other words, each simulation technique can be characterized by how much each process is allowed to advance its simulation clock beyond the current global virtual time. This is represented in Figure 2.1 [27]. The figure shows that synchronous simulation does not allow local clock advancements beyond GVT at all, while Time Warp allows local simulation clocks to advance optimistically. The Chandy-Misra algorithm allows local simulation clocks to conservatively advance based on input waiting rules, while simulation clocks advance within a given window in Moving Time Window.

2.2 Basic Data Structures

As basic data structures for logic simulation on massively parallel SIMD machines, we consider memory layout, event queue schemes, and table lookup.

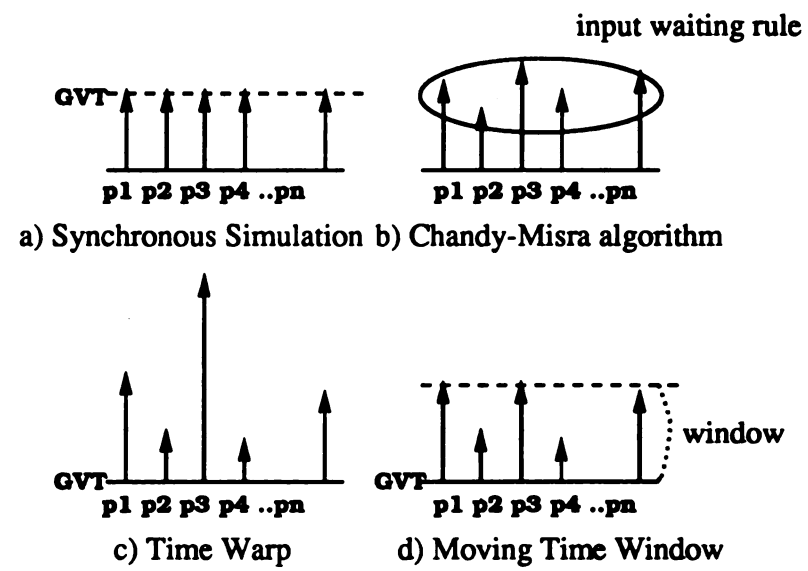


Figure 2.1: Local simulation clock advancements

2.2.1 Memory Layout for Parallel Simulation

Without loss of generality, we can assume that one gate is assigned to one processor since massively parallel SIMD machines have a large number of physical processors and allow users to define as many virtual processors as possible within available memory capacity. A process (gate) contains the following information:

MEMORY LAYOUT

- Local Virtual Time (LVT): the smallest simulation time of unprocessed events in the event queues of the process.
- Gate information: the information about the process, such as pointers to its successors.
- Function table: the table for computing output signals from input signal values.
- Input signals: the current input signals at the LVT.
- Input buffer: the buffer for storing just arrived events before putting them into its event queues.
- Event queues: the queues for storing all events to be processed.
- Other local variables.

In the above memory layout, other local variables include general temporary variables, such as variables for an active bit, an output signal, etc. as well as simulation technique-dependent variables, such as link clocks and minimum link clocks in conservative simulation. In addition, a global virtual time (GVT), the smallest timestamp of unprocessed events in all processes, is computed in the front-end processor.

2.2.2 Event Queue Scheme

In our simulation, a distributed event queue scheme is used in which each input port of a gate has its own event queue. In SIMD, the maximum queue size over all processors

inbits($b_2b_1b_0$)	000	001	010	011	100	101	110	111
3-input AND	0	0	0	0	0	0	0	1
3,2,or 1-input OR	0	1	1	1	1	1	1	1
2-input AND	0	0	0	1	x	x	x	x
1-input AND	0	1	x	x	x	x	x	x
Invert	1	0	x	x	x	x	x	x

Table 2.1: Function tables for lookup operations

should be estimated and allocated in advance since dynamic allocation is not allowed, in general. In this thesis, each gate has at most 3 input ports and 2 output ports as shown in Figure 2.2.

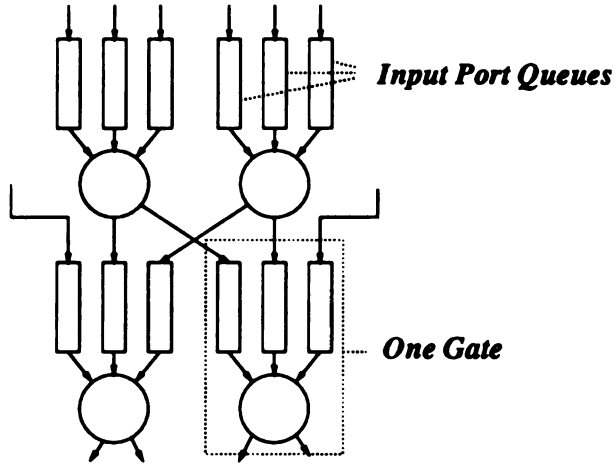


Figure 2.2: Queue assignment to a gate

2.2.3 Table Lookup

To evaluate different functions of all active gates at the same time, the table lookup method is used as in [24]. Each gate contains a table for the corresponding function. For example, if there are at most 3 inputs to each gate, 8 bits are enough to store each operation as shown in Table 2.1. In the table, "x" represents "don't care" signal.

2.3 Parallel Logic Simulation Protocols

In this section, several variations of distributed event-driven simulation protocol are considered. A logic circuit can be considered to be a directed graph in which nodes represent gates called processes, and arcs (links) indicate connections between the gates. Each gate propagates its output signals by sending event messages. Each event is time-stamped with the simulation time at which it should be executed.

Definition 1 *Simulation Cycle :* All processors on massively parallel SIMD machines are synchronized. Therefore, each process repeats the same procedure, which consists of LVT computation, choosing active gates, event evaluation, new event propagation, and queue manipulation. Optimistic simulation may require an additional step for rollback management. The time period to perform this procedure is called a ***simulation cycle.***

The diagram for possible state changes of a process is given in Figure 2.3. The idle state in the figure represents that a process is not involved in any of the above activities. Each simulation cycle is synchronized in SIMD environments.

2.3.1 Distributed Synchronous Logic Simulation

Distributed synchronous simulation is a synchronous simulation with distributed event queues. In distributed synchronous logic simulation, each input port of a process receives events in non-decreasing timestamp order. A circular FIFO queue, rather than a priority queue, can be used as an event queue of each input port, facilitating queue manipulation. During every simulation cycle, the following steps are performed.

Algorithm SYNCHRONOUS

1. Each process computes its LVT.
2. GVT is computed by taking the minimum LVT of all processes.
3. Each process whose LVT is equal to the GVT performs event evaluation and event propagation.

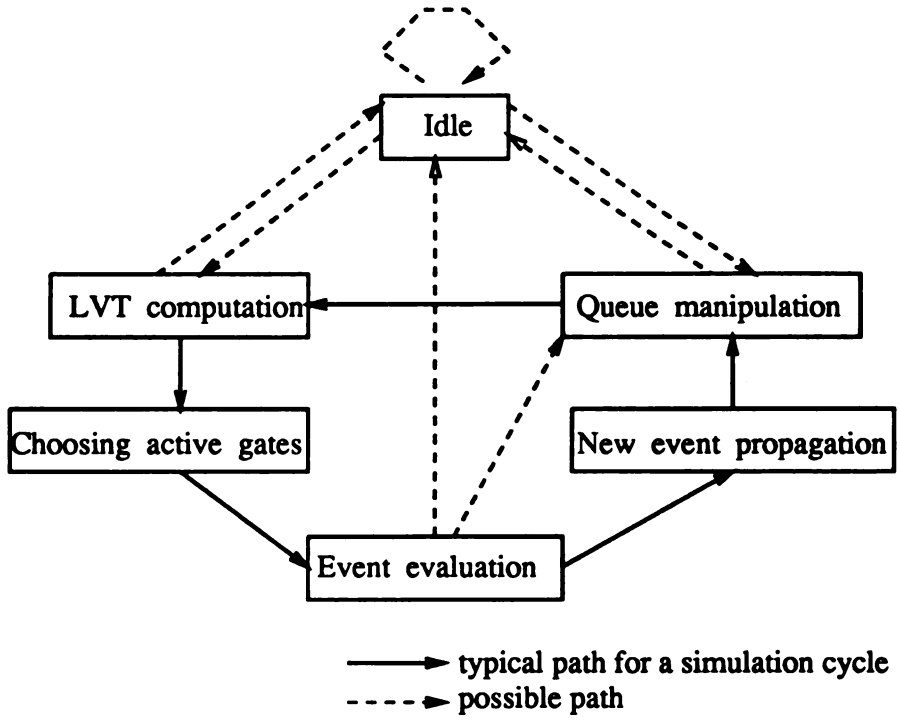


Figure 2.3: Diagram for possible state changes of a process

4. Each receiving process performs event insertion.

2.3.2 Conservative Logic Simulation

In conservative simulation, a process executes events only if it is certain that no event with an earlier timestamp can arrive. *Link clocks* and *minimum link clock* are used as defined in conservative simulation [20, 25]. A link clock of each input port of a process is defined to be the timestamp of the most recently arrived event along that input link. The minimum link clock of a process is the minimum of link clocks of all its input ports.

Since the delay of each process is fixed and there is no preemption in logic simulation, events arrive at an input link in nondecreasing order of timestamps. A circular FIFO list is assigned to each input port of a process as an event queue. The following *modified input/output waiting rules* of the Chandy-Misra algorithm are proposed which exploit gate-level logic simulation properties: **a process executes the events with timestamps equal to its LVT when its minimum link clock is greater than or equal to the LVT; the output events generated can be immediately sent to their destinations.**

Null messages in Chandy-Misra algorithms [19, 20] are used to avoid deadlock. In contrast to MIMD environments, where null messages may significantly decrease the performance of Chandy-Misra algorithms [63], the null messages in SIMD environments can be efficiently used [25] because all null messages are sent at the same time real events are propagated.

In addition, GVT can also be used to further reduce simulation cycles in conservative logic simulation. All events whose time stamps are equal to GVT are executed even though the Chandy-Misra input waiting rule is not satisfied. On a SIMD machine, GVT can be computed easily.

The proposed algorithm exploits features of the SIMD architecture: easy computation of GVT and negligible overhead of null messages. For each simulation cycle, the following steps are performed for each process:

Algorithm CONSERVATIVE

1. Each process computes its LVT.
2. GVT is computed.
3. Each process executes the events with timestamps equal to its LVT when
   a) its minimum link clock is not less than LVT or
   b) the GVT is equal to LVT;
4. Each active gate performs event propagation. Each gate which has an updated minimum link clock and does not propagate any event sends a null message with the updated minimum link clock.
5. Each receiving process sets the corresponding link clock to the timestamp of the just arrived message along the link. If the message is non-null, event insertion is performed.

There are three things to be considered to obtain good performance when we implement conservative logic simulation using null messages and GVT. First, all processes whose minimum link clocks are updated by null messages or events need to propagate new null messages or events to update link clocks of its successors in time. Second, at every simulation cycle, all link clocks should be updated to GVT if they are smaller than GVT. Finally, null messages with infinite timestamps need to be sent at the end of each input vector to inform successors that no further events will be generated.

The proposed simulation technique can also be used for logic simulation on MIMD processing environments. Since GVT computation on MIMD machines is expensive, Steps 2) and 3b) in the above algorithm might be discarded.

2.3.3 Optimistic Logic Simulation

Optimistic logic simulation uses LVT differently from conservative logic simulation. The difference is that all events whose timestamps equal LVT are executed in optimistic simulation, while the same events may not be executed in conservative simulation if the execution condition is not satisfied. Optimistic logic simulation does not need any link clocks. A simulation cycle of optimistic simulation performs the following steps:

Algorithm OPTIMISTIC

1. Each process computes its new LVT.
2. If the new LVT is not greater than its previous LVT, then rollback procedure, i.e. state restoration and cancellation, is performed.
3. Each process with unprocessed events performs event evaluation and propagation.
4. Each receiving process performs an event insertion operation.
5. Fossil collection is performed if necessary.

The following difficulties must be considered in implementing the optimistic simulation on a massively parallel SIMD machine. First, optimistic simulation has a storage overhead problem because information about all events whose timestamps are greater than GVT must be kept. Moreover, each process has three queues: input, output, and state queues. Second, processors of most massively parallel SIMD machines have relatively small local memory capacity. Finally, array access time on the CM-2, our target machine, is slow [25] because its processors are bit-serial. Therefore, efficient storage management is very important.

Immediate Cancellation

Aggressive and lazy cancellation techniques have been proposed to undo incorrect event propagation [41, 43]. However, these are difficult to implement on massively

parallel SIMD machines for the following two reasons. First, there is not enough memory space for both state and output event queues. Second, additional simulation cycles are required to send antimessages, i.e. one simulation cycle for each antimessage.

To cope with the problems, we have proposed the "immediate cancellation" technique which eliminates use of antimessages and does not require both state and output queues [29]. In this cancellation technique, a gate in rollback immediately propagates a replacement event to its successors. Immediate cancellation is a variation of aggressive cancellation which sends antimessages for all incorrect event propagation. Since logic simulation has fixed event propagation routing and no preemption, one replacement event is enough to nullify all incorrect event propagation. With the immediate cancellation scheme, we can achieve significant reduction in space and make both queue manipulation and rollback management fast and easy.

We explain the immediate cancellation technique with both aggressive and lazy cancellation techniques together where the latter two concepts were described in Chapter 1. In Figure 2.4, three cancellation techniques, immediate, aggressive, and lazy cancellation, are explained from the situation given in Figure 2.4.(a), where process C has processes D and E as its successors. Suppose that rollback occurs at process C because there is a new event with timestamp 45 when the current LVT is 60. In immediate cancellation, after the first event, which caused rollback, is processed, a rolled back process propagates a replacement event to its successors. In other words, as soon as the event evaluation finishes at the simulation time 45, a replacement event with a new output signal is propagated to both processes D and E. Here, any antimessage is not needed for the outputs already sent to both D and E at simulation times 50 and 60, since the event sent at the simulation time 45 causes a rollback at both processes D and E. The simulation continues forward again. We do not have to consider the rollback caused by the new event any more.

In aggressive cancellation, antimessages at simulation times 50 and 60 must be sent to both D and E as shown in Figure 2.4.(c), which may cause rollback at both D and E. But the antimessages are useless if a new event at simulation time 45 is

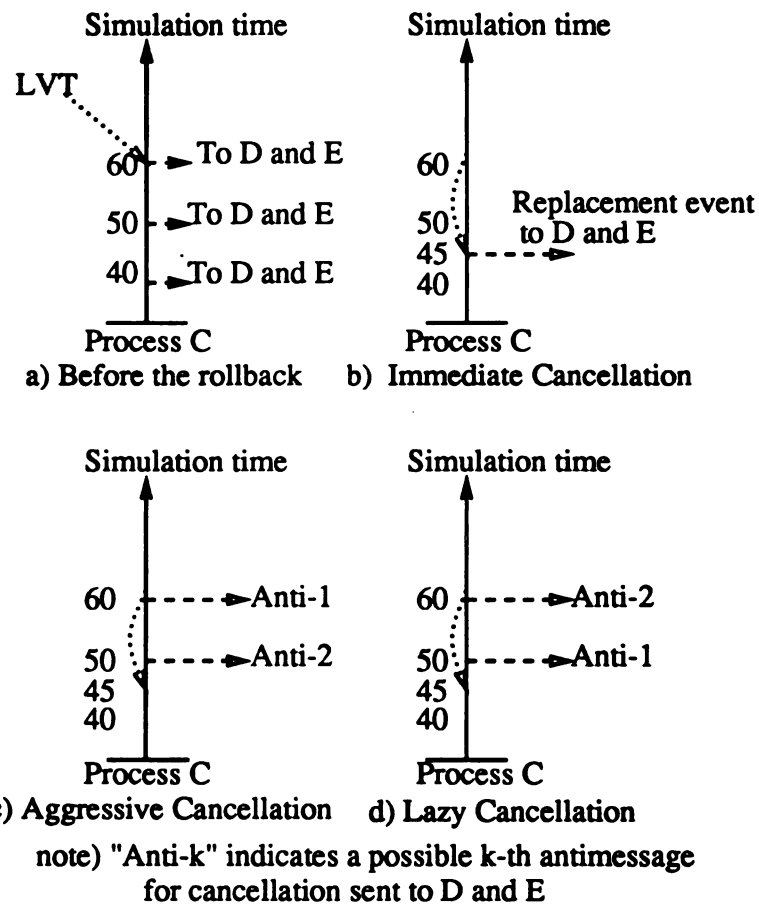


Figure 2.4: Cancellation techniques

propagated to both D and E and both the processes have to roll back again. In this case, extra time is wasted sending the useless antimessages. In addition, processes D and E also need extra time to take care of the received antimessages.

In lazy cancellation, as shown in Figure 2.4.(d), even though an event is sent at simulation time 45, antimessages at simulation times 50 and 60 may be sent. But these two antimessage can be avoided, since once a process propagates an event to its successors, the process does not have to propagate any more events (or messages) for lazy cancellation. For example, as shown in Figure 2.4.(b), if an event is propagated at simulation time 45, then we do not need to send any more messages at simulation times 50 and 60 for cancellation.

Circular Binary Search Queue

The FIFO queue structure which has been used for both synchronous simulation and conservative simulation is not suitable for optimistic logic simulation because timestamps of events along each input link are not monotonic. Several existing queue structures for optimistic simulation have been suggested: a splay tree based on a self adjusting binary tree [65]; a timing wheel using a priority queue structure [67]. In case of rollback, however, *all* the events in the queue may have to be searched to determine which to remove [14].

As an attempt to achieve fast queue manipulation, we use an event queue scheme, called CBSQ (Circular Binary Search Queue). A CBSQ is a data structure which allows a binary search on a circular list. As shown in Figure 2.5, a CBSQ has two pointers, *Front* and *Rear*, to indicate events in non-decreasing order of timestamp contained in the queue. *Front* points to the element with the smallest timestamped event, while *Rear* points to the next available element in the queue. An algorithm for finding a **key** in a CBSQ Q with n elements numbered $0 \cdots n-1$ is described as follows:

Algorithm CBSQ_search(key)

if(*Front* $<$ *Rear*) then perform binary search from *Front* to (*Rear* $-$ 1)

else if (*F*

if (

else

else

else the q

A CBS

two pointe

which has

CBSQ stru

does not n

The fol

- find a
- insert
- delete

else if ($Front > Rear$) then

if (key $< Q(n-1)$) then perform binary search from $Front$ to $(n-2)$

else if (key $> Q(n-1)$) then perform binary search from 0 to ($Rear - 1$)

else found

else the queue is empty

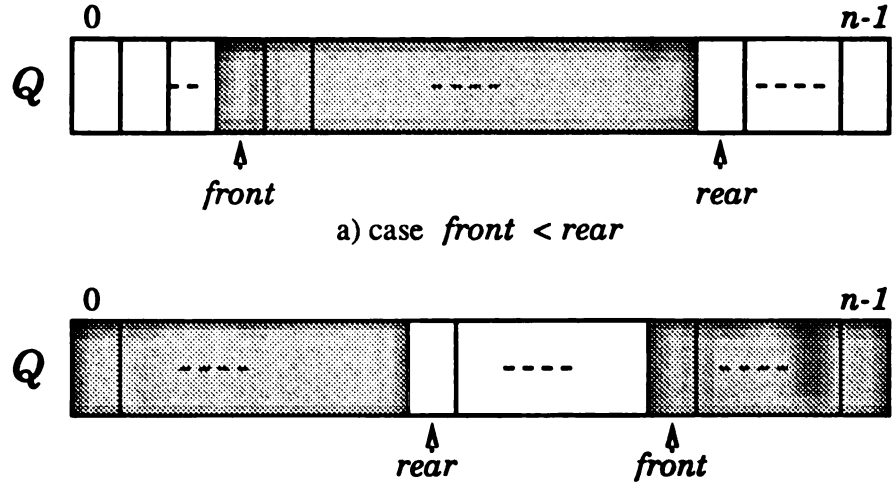


Figure 2.5: A CBSQ structure

A CBSQ is assigned to each port of a process as an event queue. In addition to the two pointers, another pointer called *SimPtr* is used to point to the unprocessed event which has the smallest timestamp and is ready to be evaluated for simulation. The CBSQ structure can be used for event queues since immediate cancellation, which does not need use of antimessages and state queues, is used.

The following basic operations can be done on CBSQs with $O(log\ n)$.

- find an event with timestamp t,
- insert an event into a queue.
- delete all events with timestamps greater than t,

- delete all events with timestamps less than t, and
- find the next event after the event pointed to by *SimPtr*.

In the above operations, all operations except the first can be done with $O(1)$ operations after the first operation is done for the same timestamp, since the first operation gives a result index from binary search. The second and third operations are needed for event insertion and elimination of all events with timestamps greater than the timestamp of a new event on the same queue. The fourth operation is used for fossil collection. Finally, the fifth operation is used to increase the pointers,*Front*, *Rear*, and *SimPtr*.

The following operations using the above basic operations are required for each simulation cycle.

- New event insertion: When a new event is inserted into a CBSQ, all the events from *Rear* to the timestamp of the new event are automatically eliminated by just moving *Rear* to the element after the new event. That is, we do not need any additional operations for deletion following rollback. If the timestamp of the new event is less than the timestamp of the event pointed to by the *SimPtr* (rollback), *SimPtr* is moved to the new event. In case of rollback, signals from events with timestamps just less than the timestamp of the new events are used to restore the state of the gate. Hence, we do not need additional state queues for state saving.

- Rollback adjustment: When the rollback occurs, CBSQs of all other inputs in the rolled back process are also adjusted. That is, *SimPtr*s are moved down to the event with timestamp equal to or just greater than the timestamp of the new LVT.

- Fossil collection: Binary search is used to find an event whose timestamp is just less than GVT, and adjust *Front* to point to that event. But fossil collection can be limited by some condition. For example, at every 10 simulation cycles, a check is made as to whether there is any queue which exceeds a predetermined

limit of queue size. If there is at least one queue which satisfies the above condition, fossil collection is performed.

On the average, it takes $O(log\ n)$ time to do a CBSQ search operation on each queue, where n is the number of events on the queue. At least two binary search operations are required on each queue for a simulation cycle. For a circuit with maximum fanin p, we need $(2+q)p$ binary search operations on the average at each simulation cycle where q is a real number ($0 \leq q \leq 1$) representing an average frequency of fossil collection. q varies depending on available queue size. For example, if we assume that unlimited sized queues are used, q is zero. If the queue size is 2^n where n is a positive integer, boundary check operations are not required. For example, for a 512 element CBSQ, 9-bit unsigned integers can be used as pointers.

2.3.4 Moving Simulation Bound(MSB)

On a massively parallel SIMD machine, a single queue overflow will halt simulation. The Moving Time Window technique has been used to control queue space overhead in Time Warp. The same technique can be applied to conservative simulation to reduce the possibility of queue space overflow.

In this thesis, for consistency, Moving Simulation Bound (MSB) refers to the bound in which active processes determine events to be executed in conservative simulation, as well as the moving time window in Time Warp. The MSB technique is a controlling mechanism for traditional event-driven simulation techniques to finish simulation successfully, not a simulation technique itself. This technique is applied when event evaluation is performed in each simulation cycle of both conservative simulation and optimistic simulation by giving a bound B as follows.

Algorithm MSB(B)

1. GVT is computed
2. Each process with new LVT is chosen as a candidate (to be active) based on event scheduling of a given simulation technique.

3. **Only each candidate process with LVT < (GVT + B) is considered an active process for simulation at every simulation cycle.**

2.4 Conclusions

This chapter studied logic simulation as one application of distributed event-driven simulation on massively parallel SIMD processing machines. Data structures to implement several simulation techniques efficiently in SIMD environments were discussed. Some variations of distributed event-driven simulation have been proposed to improve performance of logic simulation. The performance of the considered logic simulation techniques will be given in the next chapter.

Chapter 3

Performance of Parallel Logic Simulation

In this chapter, we present experimentation and performance of the logic simulation protocols considered in Chapter 2. The logic simulation protocols were implemented on the CM-2 with 32K processors and the MP-1 with 16K processors. Some ISCAS'85 and ISCAS'89 benchmark circuits are used as test circuits. The performance of the protocols are measured for 1,000 randomly generated input vectors

The remainder of this chapter is organized as follows. Section 3.1 describes the transformation of test circuits. The performance of the parallel logic simulation techniques proposed in the previous chapter is measured in Section 3.2 in terms of number of simulation cycles, parallelism, maximum queue sizes, and executions times. Section 3.3 presents the performance comparisons of logic simulation on both the CM-2 and the MP-1.

3.1 Experimentation

To get reasonable test results for performance analysis, we use three kinds of benchmark circuits: ISCAS'85 benchmark circuits [13], ISCAS'89 benchmark circuits [12], and a 32-bit array multiplier [40]. They have different characteristics as test circuits. ISCAS'85 circuits are combinational circuits, while ISCAS'89 circuits are sequential circuits which have flip-flops and feedback. In addition, a 32-bit array multiplier with 8,256 gates is used as a test circuit to compare the execution times between a VHDL simulator [1] and a distributed event-driven simulation.

Kinds	Circuit Names	Original No. of Gates	Transformed No. of Gates
ISCAS'85	C1355	546	1001
	C1908	880	1340
	C6288	2416	3880
	C7552	3513	5848
ISCAS'89	S9234	5896	7830
	S13207	8651	11860
	S15850	10470	13782
	S35932	18148	25359
Other	32bit Mul	8256	8256

Table 3.1: Test circuits used for simulation

3.1.1 Circuit Transformation

In SIMD environments, the time period taken by the slowest (or busiest) process determines the simulation cycle time since all the processors are synchronized. Maximum fanin and fanout in a circuit significantly affects the performance of logic simulation since the number of required *send* operations per cycle depends on the maximum fanin and fanout. In addition, if a gate has many input ports, a distributed event queue scheme cannot be used on the CM-2 due to the small size of local memory. For good performance, we need to transform a given circuit to be simulated into an equivalent circuit with limited fanin and fanout. The names of circuits used as testbeds and their final number of gates after the transformation are given in Table 3.1. The names of ISCAS'85 circuits start with "C", while those of ISCAS'89 start with "S". The change of circuit characteristics after transformation is shown in Table 3.2. Tree structures using dummy gates with zero-delay are used to transform gates which exceed the fanin or fanout limit. It must be guaranteed that the transformed circuit generates the same results as the original circuit.

In our simulation, the maximum fanin and fanout are 3 and 2, respectively. In this case, at most 6 (= 2x3) *send* operations are required at every simulation cycle because all combinations of input ports and output ports have to be considered during event propagation. Array multipliers do not need the transformation procedure since

Circuit Names	Before Transformation		After Transformation	
	Critical Path Length	Max (fanout,fanin)	Critical Path Length	Max (fanout,fanin)
C1355	24	12,5	36	2,3
C1908	40	16,8	60	2,3
C6288	124	16,2	174	2,2
C7552	43	15,5	55	2,3

Table 3.2: Circuit characteristics

the maximum fanin and fanout of each gate is 2. Reducing the maximum fanin and fanout also reduces both queue size and the number of queues at a gate as well as communication costs. But, as side effects, we can see that the transformed circuits have about 27 - 50 percent longer critical path lengths [10], and about 32 - 83 percent more gates than the original circuits as shown in Table 3.2. Therefore, more simulation cycles and processors are required.

3.1.2 Additional Considerations

In the simulation of sequential circuits, we need to consider global clock generation for synchronization of elements. Simulation results may be different depending on how global clocks are given to the circuit being simulated. In our simulation, without loss of generality, it is assumed that the global clock period is the same as the simulation time interval between successive input vectors. That is, whenever a new input vector enters the circuit being simulated, the next global clock is activated.

In SIMD environments, it is difficult for storage elements like D flip-flops to use clocks from a clock generator, since one send operation is required to propagate to each element. Therefore, there are two ways of handling clock generation. In the first approach, each storage element contains clock values in advance in its event queues. In this case, each clock signal is treated as an event. In the other approach, each storage element computes clock signals, which it is supposed to receive from the clock generator, based on its local virtual time. In this thesis, the second approach is used.

Let us discuss queue space based on available memory size of the CM-2. To use

aref32 and *aset32*, the fastest array access operations of the CM-2, we need to define 32 bits as the event size. For an event size of 32 bits and 1.6K bits of space for local variables, we can store up to about 1950 events, i.e. 62.4K / 32 = 1950, when a processor contains and simulates one gate only. Since each gate has three event queues, one for each input port, 1950 is divided by three. In both synchronous and conservative simulation techniques, the maximum size of a FIFO queue is 650 events. However, in optimistic simulation, the queue size is restricted to a power of 2 to avoid checking a boundary while manipulating a CBSQ. The maximum size is 512 ($=2^9$).

In this scheme, if each input gate has initial input events at port 0 only, the maximum number of input vectors to be used for simulation is the same as the event queue size. We used an event feeding technique, which feeds events to port 0 from event queues at other ports only when GVT exceeds a certain simulation time.

3.2 Performance Evaluation

Experimental results on the performance of the considered simulation protocols have been obtained for the benchmark circuits on the CM-2 with 32K processors. Although simulation with multi-delay can be done, a unit-delay was assigned to each gate for the consistency of experimental results. In the measurements, Moving Simulation Bound (MSB) is applied to both conservative simulation and optimistic simulation techniques to prevent queue overflow.

3.2.1 Performance Metrics

As performance metrics, we use the number of simulation cycles, parallelism, maximum queue size, and execution time.

Number of Simulation cycles

As defined in Section 2.3, a simulation cycle is synchronized for processors in SIMD. The execution time is proportional to the number of simulation cycles. That is, a large number of simulation cycles means slow speed. Based on the relationship between

the number of simulation cycles and execution time, we can measure the average time taken per simulation cycle.

Parallelism

The degree of parallelism (sometimes called *activity level* [63]) is the ratio of active processors to assigned processors at a given simulation cycle. An active gate at a simulation cycle is defined as a gate which has at least one event to be executed during the simulation cycle. As the number of active gates increases, the concurrency becomes higher. An active processor is defined as a processor which contains at least one active gate.

Let H be the number of assigned processors. To measure concurrency, we define degree of parallelism and parallelism as follows.

Definition 2 *Degree of parallelism (D_i)* at simulation cycle i is the ratio of the total number of active processors to the total number of assigned processors. That is, $D_i = C_i/H$, where C_i is the number of active processors at simulation cycle i.

When a processor simulates one gate only, the number of active gates is equal to the number of active processors.

Definition 3 *Parallelism* is the average ratio of the total number of active processors to the total number of assigned processors.

Parallelism can be computed as follows:

$$Parallelism = \frac{1}{S}\sum_{i=1}^{i=S} D_i$$

where S is the number of simulation cycles.

Maximum Queue Size and Fossil Collection Frequency

The maximum queue size is related to the memory requirement and speed. If there is a queue overflow, simulation cannot continue. In SIMD, the maximum queue size over

all processors should be estimated and allocated in advance. In optimistic simulation using CBSQ data structures, fossil collection is performed whenever there is at least one queue which exceeds a certain limit. The frequency of fossil collection significantly affects performance.

Execution Times

Some results in this thesis are not *ideal* (or not *general*) on massively parallel SIMD machines because some constraints of evaluation advancement were applied to limit queue size. In other words, if we had used a massively parallel SIMD machine with enough local memory size, different performance would have been obtained.

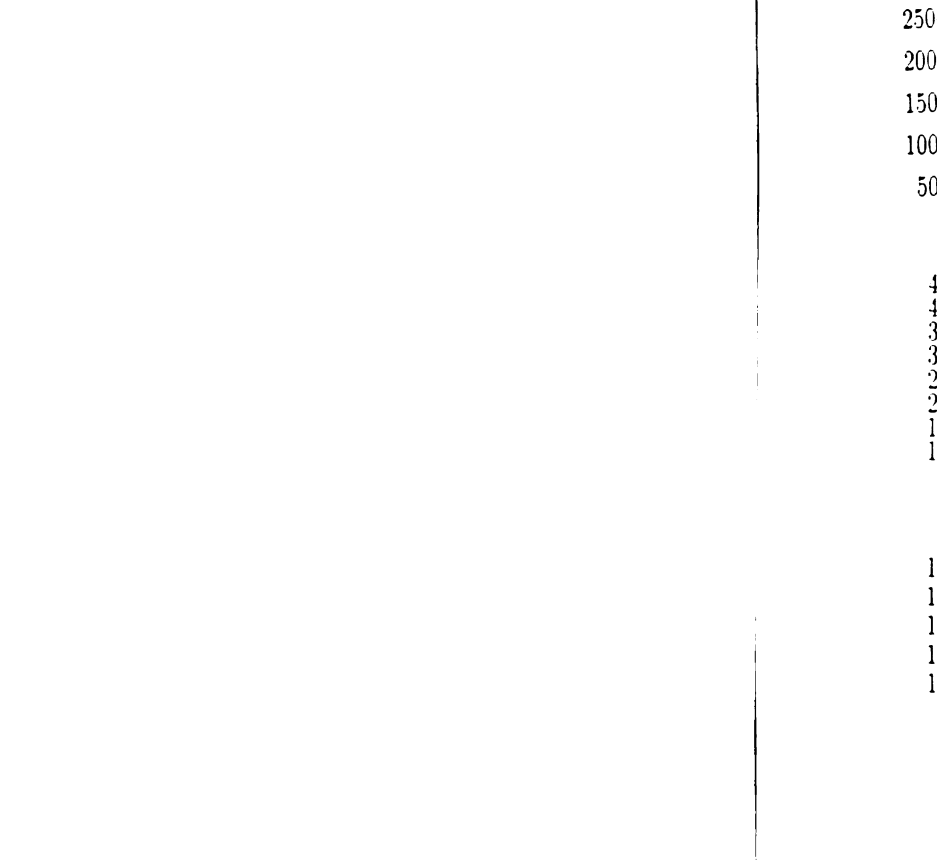
3.2.2 Experimental Results

For performance evaluation 1,000 input vectors were used. We gave 200 and 512 as timestamp intervals between successive input vectors for ISCAS'85 and ISCAS'89 benchmark circuits, respectively. The VP ratio (defined later) was 1 in the measurements.

Effect of Moving Simulation Bound

Let us first see the effect of Moving Simulation Bound (MSB) on simulation techniques. C1908 and C7552 are used as benchmark circuits to demonstrate the effect. Synchronous simulation does not require MSB since large queue size is not necessary due to event processing based on each input vector.

The effect of MSB on conservative simulation is shown in Figure 3.1. The same simulation effects were obtained for both C1908 and C7552. Let us explain the effect on C7552 in detail. When the size of MSB is greater than 10,000, queue overflow occurs. As MSB increases, the required queue size increases while both the execution time and the number of simulation cycles decrease. Here we can find an interesting fact that, for MSB $>$ 2,500, the number of simulation cycles and execution time do not change while the queue size increases. The reason can be explained as follows.



Figur

As MSB inc
fast enough
a certain M
we can find
parallel SIM
is difficult si

The perfo
ilar simulatio
effect on C1

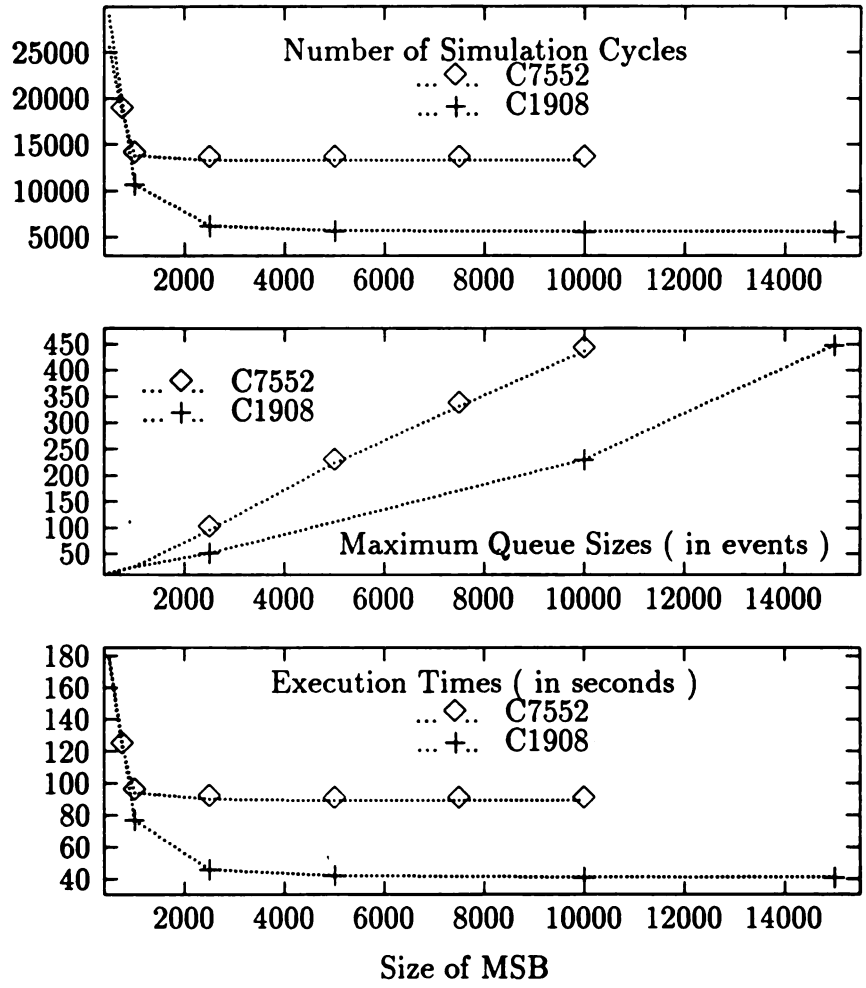
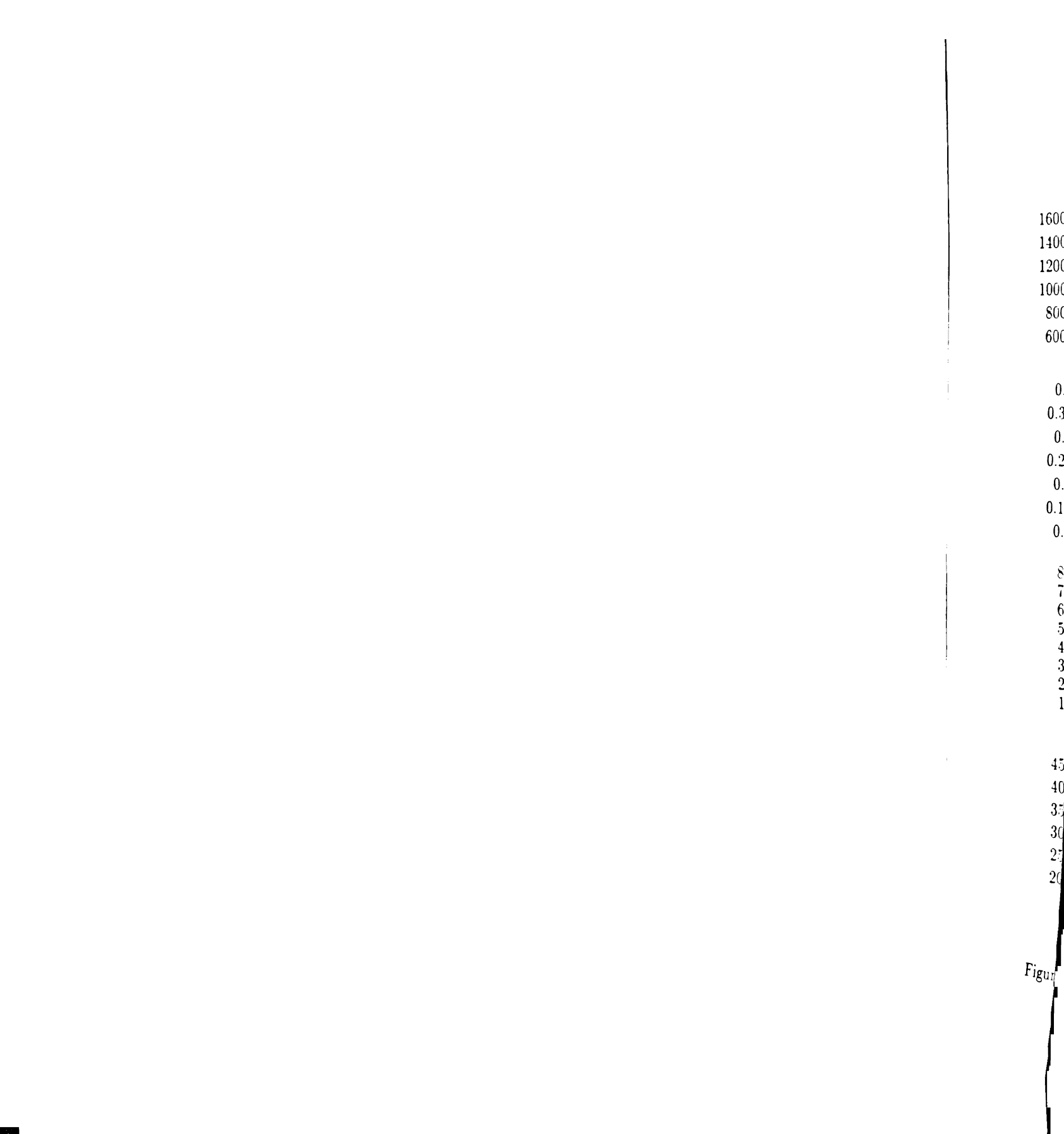


Figure 3.1: Performance based on MSB size in conservative simulation

As MSB increases, queues receive more events. But event evaluation cannot be done fast enough to consume all the events due to the input waiting rule. That is, beyond a certain MSB size, it is not necessary to increase MSB for efficient simulation. If we can find this MSB size in advance, large circuits can be simulated on massively parallel SIMD machines without any queue space problem. However, finding this size is difficult since it depends on the circuit being simulated.

The performance of optimistic simulation with MSB is given in Figure 3.2. Similar simulation effects were obtained for both C1908 and C7552. Let us explain the effect on C1908 in detail. Queue overflow occurs when MSB is greater than 18,000.


1600
1400
1200
1000
800
600
0.
0.3
0.
0.2
0.
0.1
0.
8
7
6
5
4
3
2
1
45
40
35
30
25
20

Figur

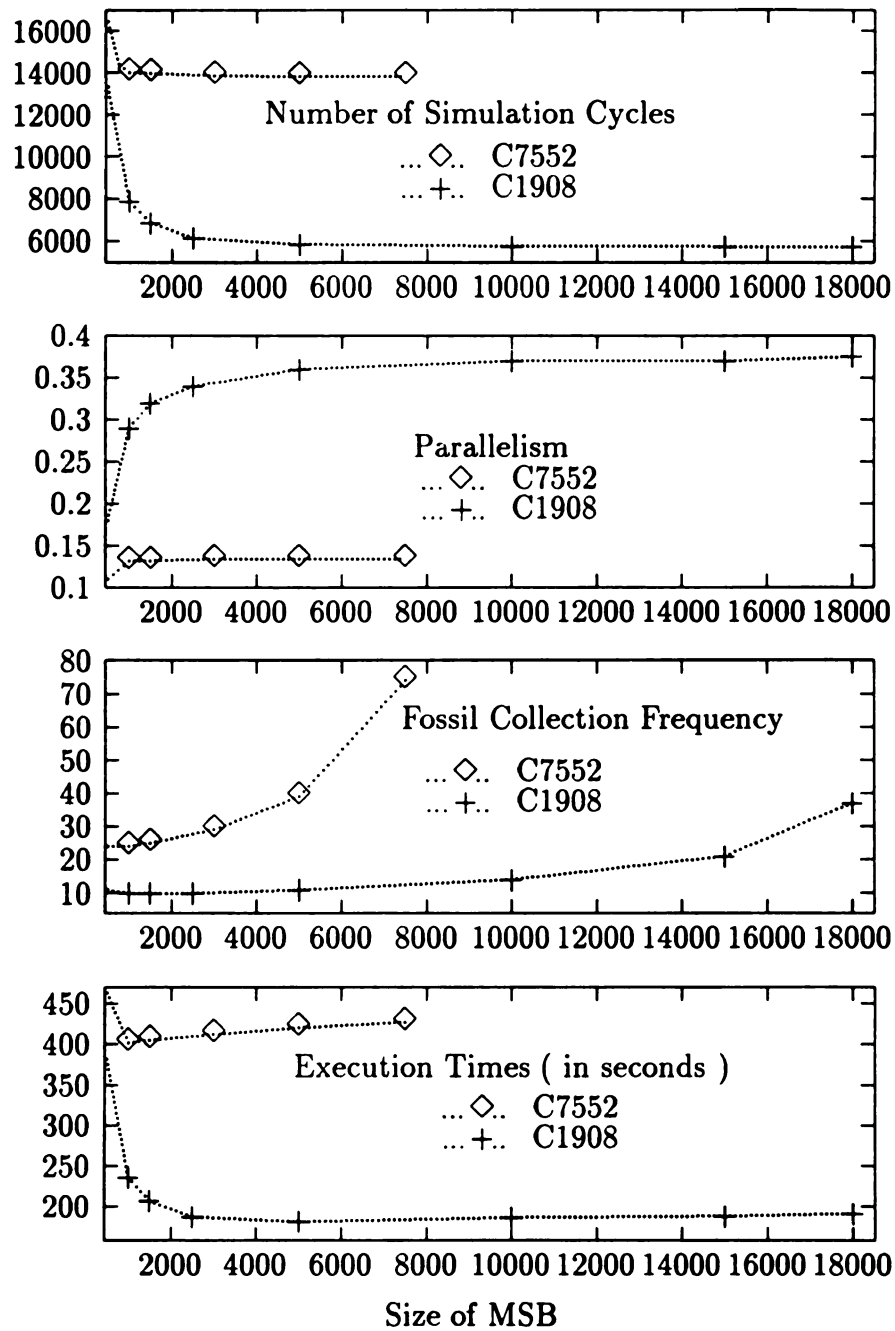


Figure 3.2: Performance based on MSB size in optimistic simulation

As the size of MSB increases, the number of simulation cycles decreases while both parallelism and fossil collection frequency increase. Optimum execution time is obtained at MSB=5,000 where the number of simulation cycles (=5855) and the fossil collection frequency (=11) are appropriately compromised. The parallelism at this point is high as well.

Number of Simulation Cycles

Table 3.3 shows the number of simulation cycles for 1,000 randomly generated input vectors. The MSB sizes to get the corresponding numbers of simulation cycles are also given in the table. According to the experimental results, synchronous simulation requires the largest number of simulation cycles, while optimistic simulation needs the least. Synchronous simulation activates only the slowest processes whose LVTs are equal to GVT at each simulation cycle. Therefore, a lot of simulation cycles are needed because a relatively small fraction of gates are involved in event evaluation. On the other hand, optimistic simulation advances the local clock of each process as far as possible. If there is an event whose timestamp is equal to or less than LVT of a gate, the gate is rollbacked immediately without loss of any simulation cycles. When rollback does not occur, the process gets as much gain as possible. In conservative simulation, event evaluation at a gate is done based on information derived from its ancestors.

In combinational circuits, synchronous simulation has many more simulation cycles than the other two techniques, while the numbers of simulation cycles for conservative simulation and optimistic simulation are close. On the other hand, optimistic simulation for sequential circuits has fewer simulation cycles than the other two techniques. The reason is that event propagation is not frequently performed in sequential circuits since flip-flops control event flow. Flip-flops execute received events according to the event scheduling policy of a technique, but they send messages only at clock times. In this case, link clocks cannot be properly updated in conservative simulation. For example, consider a gate which has a D flip-flop as its parent. In conservative simulation, the input link clock of the gate from the flip-flop is not updated in time

Circuits	Synchronous	Conservative		Optimistic	
	Cycles	Cycles	MSB	Cycles	MSB
C1355	58117	6686	∞	6678	20000
C1908	68345	5675	28000	5717	20000
C6288	177366	67658	2500	67527	1500
C7552	83784	13304	10000	13844	7500
S9234	31886	21072	∞	16310	∞
S13207	41886	30297	∞	33568	40000
S15850	58874	22794	∞	1320	∞
S35932	26062	16044	∞	1550	∞

Table 3.3: Number of simulation cycles

because local clock computation must wait until the flip-flop sends a message to the gate. In optimistic simulation, the gate can execute event evaluation as soon as possible and go ahead with virtual time advancement.

Figure 3.3 shows the number of simulation cycles for S9234 as a function of the number of input vectors. According to our experimental results, in general, optimistic simulation has the smallest number of simulation cycles, while synchronous simulation has the largest number of simulation cycles.

Parallelism

Table 3.4 shows the parallelism for 1,000 randomly generated input vectors. Optimistic simulation has the highest parallelism since it includes unnecessarily active gates which will be rolled back later. Synchronous logic simulation has the lowest parallelism. The reason is as follows. The time difference between successive input vectors is larger than the critical path length of the circuit being simulated. Since only the gates with the smallest LVTs are involved in event evaluation, any two successive vectors cannot be overlapped. In other words, input vectors are processed one by one. Therefore, parallelism is very low.

Combinational circuits have higher parallelism than sequential circuits since flip-flops in sequential circuit control (or reduce) the flow of events. We can see, in the table, that parallelism becomes low if MSB is used.

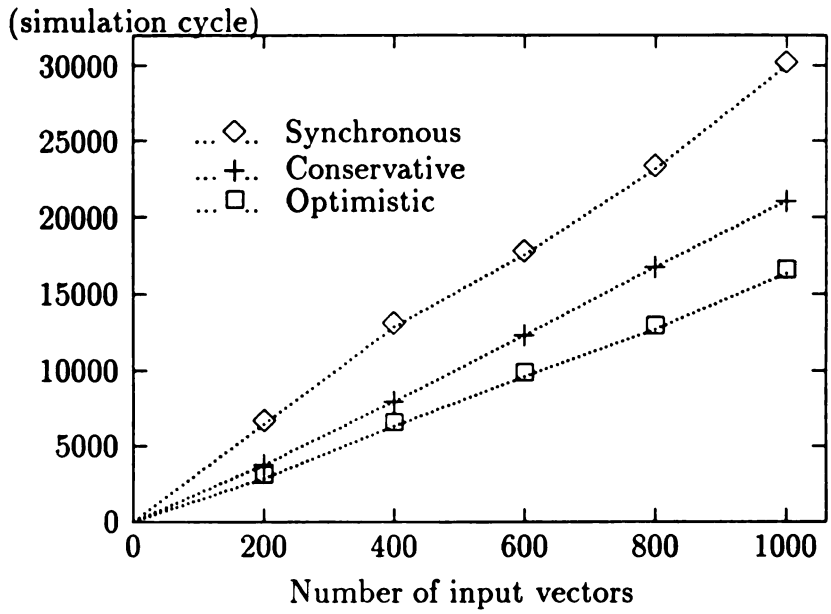


Figure 3.3: Number of simulation cycles for S9234

Figure 3.4 shows the degree of parallelism in three simulation techniques for C1355 with 200 input vectors. Synchronous simulation, conservative simulation, and optimistic simulation have 11525 (0.023), 1332 (0.19), and 1328 (0.29) simulation cycles (parallelism), respectively. The figure shows the degree of parallelism for the first 1400 simulation cycles of synchronous simulation. Synchronous simulation has a low waveform in the degree of parallelism due to its low activity level. Conservative simulation and optimistic simulation in the figure follow the trend shown in Table 3.4.

A higher level of parallelism does not automatically imply better speed. For example, in optimistic simulation there are some active gates which will be rolled back later. For a simulation technique with a given circuit, however, high parallelism indicates that good performance can be obtained as the circuit size increases.

Figure 3.4 shows that, in both optimistic simulation and conservative simulation, highest parallelism will occur when the number of simulation cycles is less than or equal to the input vector size plus the critical path length of the circuit being simulated. But parallelism becomes low as the simulation continues.

Circuits	Synchronous	Conservative		Optimistic	
	Parallelism	Parallelism	MSB	Parallelism	MSB
C1355	0.0230	0.1900	∞	0.324	20000
C1908	0.0230	0.2791	28000	0.380	20000
C6288	0.0860	0.1976	2500	0.260	1500
C7552	0.0160	0.1053	10000	0.133	7500
S9234	0.0020	0.0033	∞	0.024	∞
S13207	0.0018	0.0021	∞	0.0068	40000
S15850	0.0024	0.0027	∞	0.049	∞
S35932	0.0140	0.0230	∞	0.24	∞

Table 3.4: Parallelism

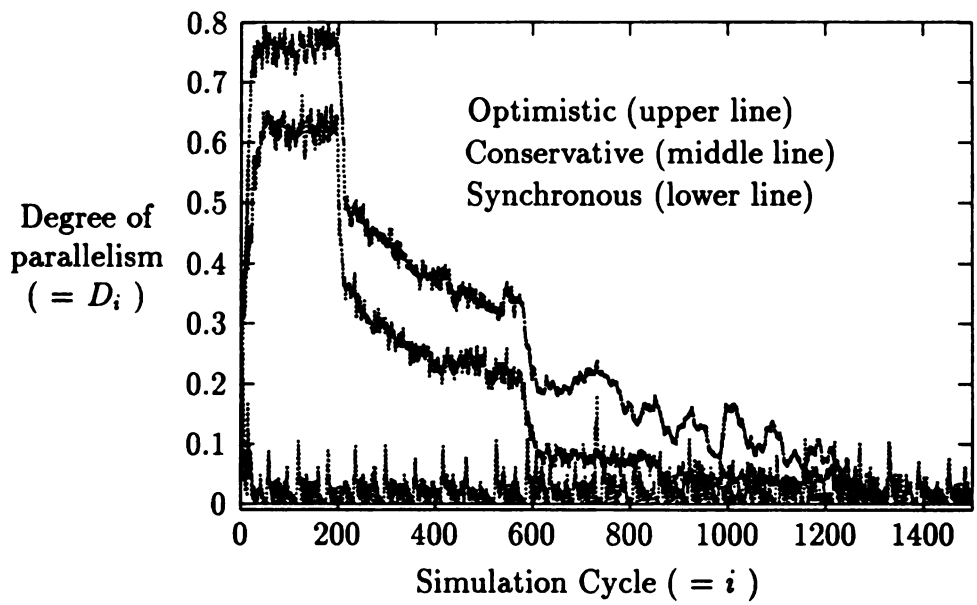


Figure 3.4: Degree of parallelism

Circuits	Synchronous	Conservative		Optimistic	
	Q Size	Q Size	MSB	Q Size	MSB
C1355	2	647	∞	346	20000
C1908	2	630	28000	473	20000
C6288	2	632	2500	419	1500
C7552	2	436	10000	431	7500
S9234	2	427	∞	337	∞
S13207	2	284	∞	434	40000
S15850	2	386	∞	258	∞
S35932	2	271	∞	289	∞

Table 3.5: Maximum queue size (in events)

Queue Size

Table 3.5 presents the required queue size of a gate at the possible maximum MSB size for each test circuit with different simulation techniques. The measurement of the maximum queue size considers all event queues during the entire simulation. The smallest queue size is required in synchronous simulation compared with the other two techniques since parallelism is low and simulation is done for input vectors one by one. On the other hand, optimistic simulation needs a large maximum queue size since events must be stored in the queue to cope with rollback. In sequential circuits, however, conservative simulation might need larger queues than optimistic simulation since conservative simulation must wait to execute events due to a rare event (or message) propagation.

In actual simulation, to avoid frequent fossil collection and improve performance, the following strategy can be used: **fossil collection is performed if the queue size exceeds the limit**. But the experimental results in Table 3.5 were obtained by applying fossil collection at each simulation cycle.

Execution Times

Table 3.6 compares the execution times of parallel logic simulation techniques for different test circuits with 1,000 randomly generated input vectors. In the measurement,

only the time period for simulation is considered. The figures listed do not include the time required for translating circuit description, reading vectors, or printing output.

We observed that conservative simulations are faster than synchronous and optimistic simulations for combinational circuits even though optimistic simulation requires fewer simulation cycles. One reason is that, each cycle of optimistic simulation involves more time-consuming operations, such as rollback and long queue manipulation. This observation contrasts with the claims in [34, 48] that Time Warp outperforms the Chandy-Misra algorithms. But, as shown in the experimental results, optimistic simulation may be better than any other techniques for sequential circuits. But, for circuits with fewer flip-flops, such as S9234, Time Warp might be slow because optimistic logic simulation has only slightly fewer simulation cycles than conservative logic simulation. Even though each simulation cycle of synchronous simulation is very simple, it usually takes more time than the two techniques since it needs many more simulation cycles.

Soule and Gupta [63] found that deadlock avoidance with null messages on the Encore Multimax (shared memory MIMD machine with 16 nodes) is highly inefficient. They claimed that actual run times range from 30 times slower to more than 100 times slower than conservative simulation with deadlock detection and recovery. But conservative simulation with deadlock detection and recovery is difficult to implement on SIMD machines. As we can see from the table, conservative simulation with null messages works very fast.

Comparison of execution times for a 32-bit array multiplier with 8256 gates between Intermetrics VHDL simulator on a SUN3/260 and conservative simulation on the CM-2 is shown in Table 3.7. All the schemes considered in this thesis are also considerably faster (up to several hundred times) than the Intermetrics VHDL simulator [1] which is a general purpose behavioral simulator.

Circuits	Synchronous	Conservative		Optimistic	
	Times	Times	MSB	Times	MSB
C1355	214	55	∞	189	20000
C1908	265	41	28000	184	20000
C6288	735	437	2500	1667	1500
C7552	324	89	10000	402	7500
S9234	119	100	∞	443	∞
S13207	154	142	∞	814	40000
S15850	227	111	∞	39	∞
S35932	100	80	∞	37	∞

Table 3.6: Execution times (in seconds)

No. of input vectors	2	4	6	8	300	500
VHDL	203.3	420.2	700.2	1023.2	-	-
Conservative Sim.	1.5	2.4	3.0	3.6	90.4	123.9

Table 3.7: Execution times (in seconds) for a 32-bit array multiplier

3.3 Comparisons of Massively Parallel Machines

Architectures of the CM-2 and the MP-1 are compared for parallel logic simulation. In view of logic simulation, the critical features of a SIMD machine are the number of processors and the memory per processor. The CM-2 is known as a parallel machine with a large number of small processors, while the MP-1 has a large processor memory size relative to other SIMD machines. We discuss machine-dependent characteristics in logic simulations on the two SIMD machines.

- **Performance of Related Instructions**

  The performance of the major instructions related to logic simulation is measured. MPL(Massively Parallel Language) is used for the measurement on the MP-1, while C/Paris is used on the CM-2. By knowing the execution time of each instruction used in our implementation, we can estimate the performance

Instruction	Global Router		Global Min		Array Write		Array Read	
in bits	32	64	32	64	32	64	32	64
CM-2	960	1600	240	470	40	80	70	140
MP-1	422	562	92	142	12	13	12	13

Table 3.8: Performance of instructions (in microseconds)

of the simulation protocols and use this information to find better data structures and algorithms for the protocols by avoiding the most time-consuming instructions. The ratio of the number of virtual processors to physical processors is referred to as the *VP (Virtual Processor) ratio.* The VP ratio was 1 in these measurements. The performance of the instructions on the CM-2 and MP-1 will be given Table 3.8 [23]. In the table, non-conflict send operations are considered in the measurement. According to the measurements, the MP-1 instructions execute 2 to 10 times faster than the CM-2.

- **Queue Space**

  One of the important factors for efficient parallel logic simulation is to have enough memory space for event queues. Each processor on the MP-1 and the CM-2 has 16K and 8K bytes of local memory, respectively. Moving simulation bounds can be used to prevent overflow in event queues for conservative simulation and optimistic simulation.

- **Gate to Processor Ratio**

  The number of available processors is another significant factor when each processor is assigned to a gate. The CM-2 provides up to 64K physical processors, and offers the virtual processor concept for circuits larger than 64K gates, which indicates how many times each physical processor must perform a certain task in order to simulate the appropriate number of virtual processors [66].

  The MP-1 has fewer processors than the CM-2. The number of physical processors of the MP-1 family ranges from 8K to 16K. Using the large local memory,

the virtual processor concept can also be implemented, although it must be explicitly written as part of the program.

- **Communication Scheme**

  The CM-2 [66] uses a Hypercube interprocessor communication topology, while MasPar's topology is a 2D mesh and external router. Each CM-2 processor chip contains one router node, which serves the 16 data processors on the chip. The router nodes on all the processor chips are wired together to form the complete router network. The algorithm used by the router can be broken into stages called *petit cycles*. The delivery of all the messages for a *send* operation might require only one petit cycle if fewer processors are active, but, if every processor is active, then many petit cycles are typically required.

  In MasPar, the Global Router is a bidirectional communication path and a circuit switched style network organized as a 3 stage hierarchy of crossbar switches [11]. Each square matrix of 16 processors in the processor array is called a *cluster*. The system can communicate with all PE clusters simultaneously, but it can communicate with only one processor per cluster at one time [51].

For the comparison of performance on the machines, a term is defined as follows.

Definition 4 *Speed ratio* is defined as the ratio of execution time on the CM-2 to that on the MP-1.

Table 3.9 shows the results of the performance of synchronous, conservative, and optimistic logic simulation for 1,000 randomly generated input vectors [23]. In the measurements, the same MSB size and input vectors which were used in Section 3.2.2 are applied to prevent queue overflow. The MP-1 runs 2 to 2.5 times faster. The obtained results are reasonable when we consider the speed of related instructions in Table 3.8.

As you can see, the optimistic protocol gives more advantages than synchronous and conservative protocols. We can consider two analytical factors, array access frequency and congestion. The first factor, array access frequency, produces very

Benchmark Circuits	Synchronous	Conservative	Optimistic
C1355	1.89	2.12	2.52
C1908	1.84	1.95	2.36
C6288	1.89	1.92	2.52
C7552	1.86	2.07	2.48

Table 3.9: Speed ratios of simulation protocols

manifest effects and wide differences. In optimistic simulation, a lot of event searching causes frequent array accesses in each simulation cycle. As shown in 3.8, the array access time is much faster on the MP-1 than the CM-2. The performance of optimistic simulation on the MP-1 is almost two and half times better than that on the CM-2.

Thus, for optimistic simulation, the performance difference for queue manipulations dominates the difference resulting from high congestion. The performance of conservative simulation on the MP-1 is also affected by this domination, producing a higher speed ratio than synchronous simulation. Thus, we can see the congestion problem does not have an important effect in ISCAS'85 circuits. The magnitude of the congestion problem, however, depends on how gates are allocated to processors.

Based on the experimental results, several factors have been considered for the analysis of the two machines. First, as the number of propagation events increases, the MP-1 has a more serious congestion problem than the CM-2 since the MP-1 uses an external global router. Among the three logic simulation approaches, optimistic simulation has the heaviest congestion overhead because it requires many propagation events. This problem can be avoided by assigning gates to processors properly. Second, the MP-1 provides bigger local memory size than the CM-2. So, the MP-1 is good for simulation protocols which require large queue sizes, such as optimistic simulation. Third, due to faster array access of the MP-1, optimistic simulation, which requires a lot of event handling, can get good performance. Finally, the CM-2 has a larger number of processors. Therefore, a circuit with a large number of gates can be simulated on the CM-2 with VP ratio 1.

3.4 Conclusions

We have experimentally analyzed the effect on performance of logic simulation depending on several factors, such as target machine used, simulation technique applied, event queue structures implemented, and test circuit simulated.

The performance of logic simulation also depends on the speed of several instructions. These related instructions are faster on the MP-1 than the CM-2. Three logic simulation algorithms are also evaluated and compared on both machines. Experimental results show that the MP-1 is about 2 to 2.5 times faster than the CM-2 for all three simulation techniques.

We observed that, despite theoretical arguments to the contrary, optimistic simulation such as Time Warp is not the best technique for *all* applications on massively parallel SIMD machines. This is attributed to its inherent rollback and queue management overhead. We also observed that, in contrast to MIMD environments, conservative simulation with null messages works very fast on massively parallel SIMD machines. Finally, we conclude massively parallel SIMD machines can be efficiently used for parallel logic simulation if we utilize the limited local memory efficiently.

Chapter 4

Performance Prediction Based on Gate-to-Processor Ratio

One of the critiques of parallel logic simulation on massively parallel SIMD machines is that the parallelism is low. In this chapter, we show that the parallelism on massively parallel SIMD machines can be increased if we increase the gate-to-processor ratio (ratio of the number of gates to the number of processors). A probabilistic model is proposed to estimate the performance of parallel logic simulation based on the gate-to-processor ratio. Using this model, parallelism, number of simulation cycles, and execution times are predicted as the gate-to-processor ratio increases. To compare the predicted performance estimation with experimental results, three distributed simulation protocols for logic simulation, synchronous simulation, conservative simulation, and optimistic simulation, were implemented. The model generally predicts more improvement in performance than experimental results indicate. We found out that the correlation between the predicted performance and the experimental results depends on both simulation protocols applied and circuits simulated. The correlation is best for optimistic simulation, and worst for conservative simulation.

The remainder of this chapter is organized as follows. Section 4.1 explains why performance estimation at various gate-to-processor ratios is needed. In Section 4.2, a probabilistic model for estimating performance based on the gate-to-processor ratio is proposed. Parallelism, number of simulation cycles, and execution times are predicted using the model in Section 4.3. In Section 4.4, the experimental results are compared with the predicted performance.

4.1 Introduction

It may be necessary to simulate a circuit with an enormous number of gates, while the number of available processors on a particular machine is limited. In MIMD environments, many gates are usually assigned to a processor since the number of processors is relatively small (compared to massively parallel SIMD machines) and each processor has powerful processing capabilities. But, even though massively parallel SIMD machines have a lot of processors, we may need to assign more than one gate to a processor to accommodate large circuits. Currently, it is difficult to do simulation with multiple gates in a processor, because existing massively parallel SIMD machines do not have enough local memory. According to manufacturer announcements [53], machines with large local memories will be released in the future. However, we can use the currently available machines by controlling event queue sizes with some moving simulation bound, such as MTW (Moving Time Window) [61]. In this case, in general, more simulation cycles and longer execution times are required.

As long as the memory capacity of a processor allows, we may assign many gates to a processor to increase the efficiency (or utilization) of the machine even though we have enough processors available. In this case, the performance can be predicted using a probabilistic model. In this thesis, how the ratio of the number of gates to the number of processors affects the performance of logic simulation on massively parallel SIMD machines will be investigated based on a probabilistic model and experimental results.

4.2 The Model

In [3], Agrawal and Chakradhar presented a statistical model of parallel processing for evaluating the performance of several synchronized iterative algorithms on multiprocessor systems. Based on their model, the speedup was estimated as the number of processors increases. Logic simulation of several VLSI circuits was used as a test problem to get experimental results.

We propose a model for analyzing the performance of parallel logic simulation on massively parallel SIMD machines. Our performance model uses the same assumptions as those in [3]. Based on our model, parallelism and the number of simulation cycles are estimated as the gate-to-processor ratio increases. Finally, we can estimate the execution time for a simulation using a known cycle time.

An *active gate* at a simulation cycle is defined as a gate which has at least one event to simulate during the simulation cycle and is ready for event evaluation. In our simulation model, we use the following model.

- Gates are assumed to be statically distributed among processors.
- Approximately the same number of gates are assigned to each processor.
- All the physical processors are synchronized and compute independently in between the successive simulation cycles.
- The amount of work performed on an active gate during a simulation cycle is called an *atom*. All atoms are statistically independent and have the same level of activity.
- The probability of a gate being active is independent of the gate-to-processor ratio.
- The active probability in steady state is constant.

Let n_i denote the number of active gates among the N_i gates which belong to processor i. Then n_i is a random variable that can only assume the values $0, 1, 2, ..., N_i$. The number of active gates within a physical processor during a particular simulation cycle is a random variable with a binomial distribution. Thus, the probability that a processor has x active gates is computed as follows.

$$P[n_i = x] = \begin{pmatrix} N_i \\ x \end{pmatrix} p^x (1-p)^{N_i - x} \tag{4.1}$$

where p is the probability that a particular gate is active.

4.3 Performance Estimation

Based on the probabilistic model proposed in the previous section, performance with respect to parallelism, number of simulation cycles, and execution times will be estimated in this section.

4.3.1 Parallelism

Let H and N be the number of assigned physical processors and the number of gates, respectively. Gate-to-processor ratio and parallelism are defined as follows.

Definition 1 *Gate-to-processor ratio (GP ratio* or *g)* is defined as the ratio of the total number of gates to the number of assigned processors. In this case, $g = \lceil N/H \rceil$. For simplicity, it is assumed that gates are equally distributed among processors.

Definition 2 *Parallelism (P_g)* is defined as the ratio of the average number of active processors to the number of assigned processors at GP ratio g.

The symbol $E(P_g)$ denotes the *expected value* of P_g.

Theorem 1 *Suppose that only one of the active gates in a processor is involved in event processing during a simulation cycle. $E(P_g)$ is $1 - (1-p)^g$, where p is the probability that a particular gate is active.*

Proof: Suppose we have H processors. Each processor has g gates. From Equation 4.1 in the model of Section 4.2, the number n_i of active gates in processor i is a binomial random variable. A processor becomes active if the processor has at least one active gate. Therefore, the probability that processor i is active is $P[n_i \geq 1]$.

Since all processors were assumed to compute independently, parallelism at GP ratio g is computed as follows:

$$P_g = \frac{\sum_{i=1}^{i=H} P[n_i \geq 1]}{H}$$

Therefore, the expected value of P_g is

$$E(P_g) = E(\frac{\sum_{i=1}^{i=H} P[n_i \geq 1]}{H})$$

$$= P[n_i \geq 1]$$

$$= 1 - P[n_i = 0]$$

$$= 1 - \begin{pmatrix} g \\ 0 \end{pmatrix} (1-p)^g$$

$$= 1 - (1-p)^g$$

□

We can get $E(P_1) = p$ from the model. To estimate $E(P_g)$, we can use $p = P_1$ as a special case.

If a processor evaluates events of all active gates during each simulation cycle, the number of active gates may be different from processor to processor since static allocation is assumed. We get the following corollary:

Corollary 1 *When a processor executes events of all active gates during each simulation cycle,* $E(P_g)$ *is also* $1 - (1-p)^g$

Proof: We already assumed that in steady state the active probability p is constant regardless of GP ratio. As the proof in Theorem 1, a processor is active if it has at least one active gate. □

But, the case that only one gate is involved in event evaluation is more efficient than the case that all gates in a processor are involved during each simulation cycle. The first case gives better performance on massively parallel SIMD machines. In the second case, a simulation cycle time is the longest processing period of all processors since all processors must wait until the slowest processor finishes the cycle. That is, the load balancing problem would be more severe during a simulation cycle because the number of active gates ranges from 0 to g.

We implemented parallel logic simulation using the three simulation protocols on the CM-2 and measured the performance in Chapter 3. In the measurement, GP

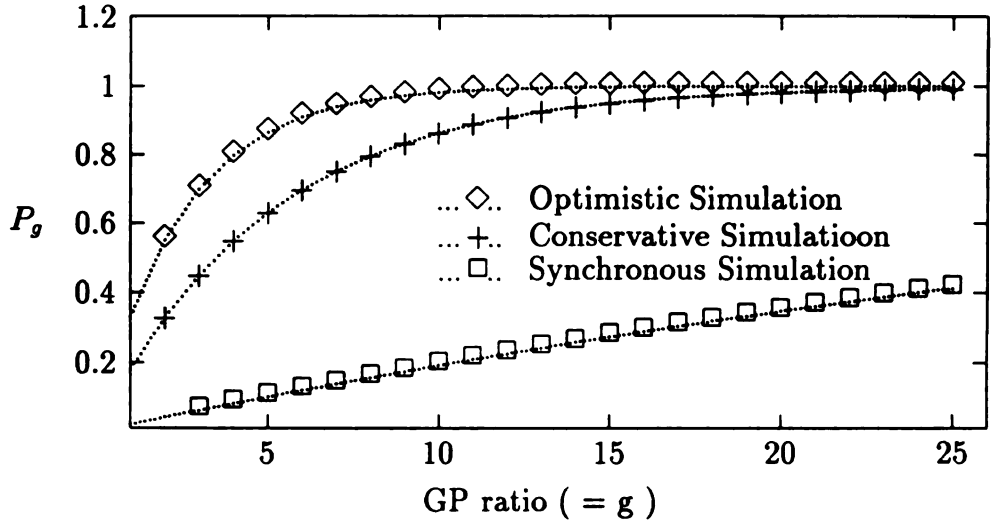


Figure 4.1: Parallelism depending on GP ratio

ratio was 1, i.e. a processor contains one gate only. ISCAS'85 circuits were used as test circuits. According to experimental results, synchronous logic simulation, conservative logic simulation, and optimistic logic simulation with no simulation time bound have average active ratio 0.021, 0.18, and 0.33, respectively. We can use these values in our model to predict the parallelism for other GP ratios. Figure 4.1 shows the predicted parallelism of three logic simulation protocols for GP ratio up to 25.

4.3.2 Number of Simulation Cycles

The number of simulation cycles is very important since it is proportional to the execution time. In this section, the number of required simulation cycles at a certain GP ratio is computed based on the proposed model.

Let W_g be the amount of work, i.e. the total number of atoms, for a simulation at GP ratio g. For the prediction of number of simulation cycles, we need to use the following assumption.

Assumption 1 $W_1 = W_g$ for any g

The assumption means that the total amount of work to be done is the same

regardless of GP ratio g.

Let S_g be the number of simulation cycles for a simulation at GP ratio g. The following lemma can be obtained:

Lemma 1 *S_g is bounded as follows:*

$$S_1 \leq S_g \leq gS_1$$

Proof: Proof is done by induction on g. Basis is established when $g = 1$. Suppose that the inequality is true for g. We must prove that the inequality is true for $g+1$. If we increase g by 1, each processor contains one more gate. Therefore, in the worst case, i.e. all gates at GP ratio 1 are active during simulation, the number of simulation cycles is increased by S_1 according to Assumption 1.

Therefore

$$S_1 \leq S_{g+1} \leq S_g + S_1 \leq gS_1 + S_1 = S_1(g+1)$$

□

We use P_g and $E(P_g)$ interchangeably throughout the remainder of this chapter.

Theorem 2 *Suppose that only one active gate is involved in event processing during a simulation cycle. When $P_1 = p$ is given for a simulation, S_g is pgS_1/P_g.*

Proof: Let N be the number of gates to be simulated. We can compute W_1 and W_g as follows:

$W_1 = S_1 Np$ and

$W_g = S_g(N/g)P_g$.

From Assumption 1, we get $W_1 = W_g$ for any g. Thus,

$$S_1 Np = S_g(N/g)P_g$$

From the equation, we can get the following solution.

$$S_g = pgS_1/P_g$$

□

Corollary 2 *Suppose that all active gates in a processor are involved in event evaluation during a simulation cycle. In this case, S_g for any g is S_1.*

Proof: All active gates are determined at the beginning of each simulation cycle. That is, any active gate does not make another gate active by sending events during a simulation cycle. The amount of work done during each simulation cycle at GP ratio g is the same as that done in the corresponding simulation cycle at GP ratio 1. Therefore, the total number of simulation cycles is S_1. □

To see the change of the number of simulation cycles with a simulation protocol as GP ratio increases, we define a new term as follows.

Definition 3 *Cycle increase ratio (R_g)* is defined as the ratio of the number of simulation cycles at GP ratio g compared with GP ratio 1, i.e. $R_g = S_g/S_1$

Note that, from Theorem 2,

$$R_g = S_g/S_1 = pg/P_g$$

Figure 4.2 shows R_g for each simulation protocol as a function of GP ratio. For a simulation protocol with high active probability, such as optimistic simulation, the cycle increase ratio increases rapidly.

4.3.3 Execution Times

The execution time for a simulation is predicted if we know the number of simulation cycles and the time taken for a simulation cycle. If the time taken for a simulation cycle, denoted by τ, is assumed to be independent of GP ratio, the execution time at GP ratio g, E_g, is computed as

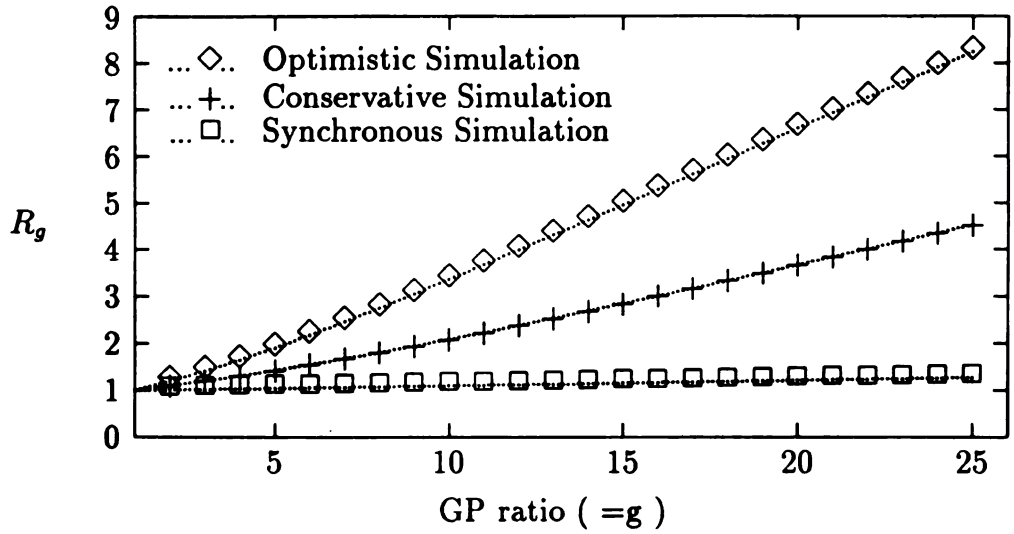


Figure 4.2: Increase ratio of simulation cycles

$$E_g = S_g \times \tau = (pgS_1/P_g) \times \tau = pg\tau S_1/P_g$$

Let us estimate the execution times of three different simulation protocols on benchmark circuits, C1355, C1908, S9234, and S35932. All the data for the estimation was obtained from the experimental results in Chapter 3. For example, the time taken for a simulation cycle is equal to execution time divided by the number of simulation cycles. But, we could not get the data for C1908 since MSB was used to prevent queue overflow. So, we wrote programs and ran them on a SUN workstation to get parallelism and the number of simulation cycles without using MSB.

The predicted execution times as a function of GP ratio are shown in Figures 4.3 and 4.4. According to the figure, as GP ratio increases, in general, optimistic simulation gives worse performance than the other two protocols because it has a higher active ratio and requires more time per simulation cycle. As a good example, we can observe that optimistic simulation for S35932 is faster than the other two simulation techniques when GP ratio is small as in Figure 4.4. However, as the GP ratio increases, the predicted execution times increase drastically, since parallelism is much higher than the other two techniques.

To analyze the decrease of speed as the GP ratio increases, we introduce a new term as follows.

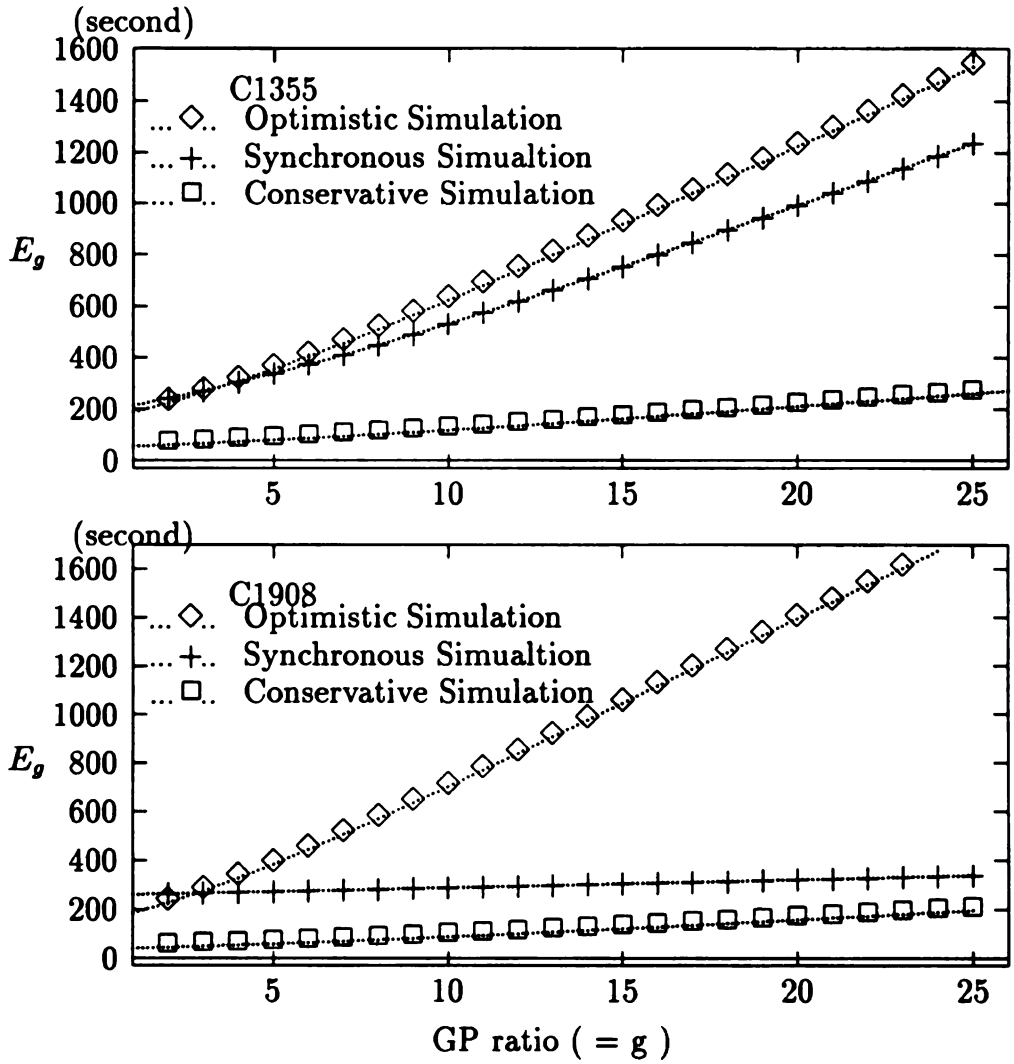


Figure 4.3: Predicted execution times for C1355 and C1908

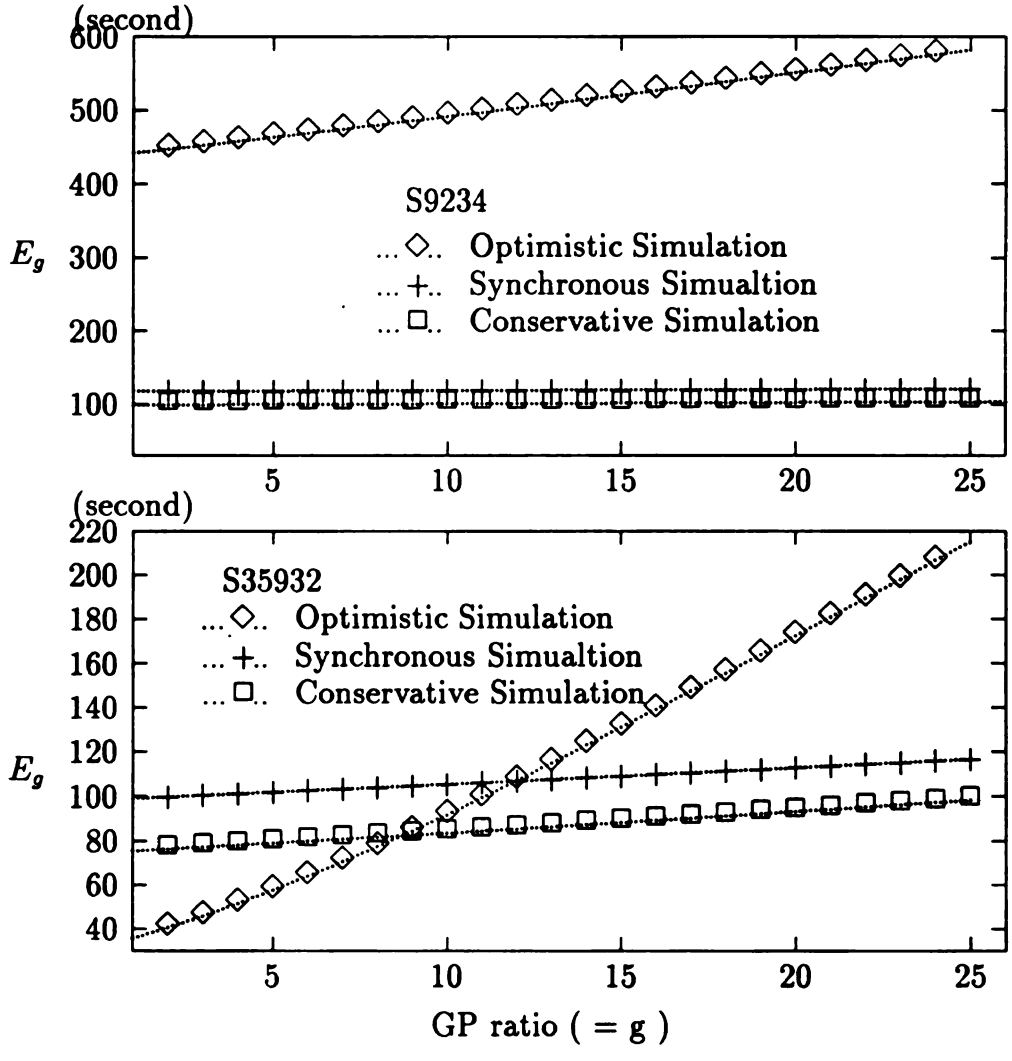


Figure 4.4: Predicted execution times for S9234 and S35932

Definition 4 *Speeddown (O_g)* is defined as the increased ratio of the execution time at GP ratio g compared with E_1, *i.e.* $O_g = E_g/E_1$

Theorem 3 *For a simulation,* $O_g = R_g$

Proof:

$O_g = E_g/E_1 = \tau S_g/\tau S_1 = S_g/S_1$.

By Theorem 2,

$S_g = pgS_1/P_g$

Therefore,

$O_g = (pgS_1/P_g)/S_1 = pg/P_g = R_g$.

□

In the worst case, speeddown is g. It is very important to maximize the difference between g and actual speeddown.

4.4 Comparisons with Experimental Results

Since the currently available massively parallel SIMD machines do not have enough local memory to run logic simulation with a high GP ratio, sequential programs were written in C and run on SUN workstations. From the programs, we can obtain the number of simulation cycles and parallelism, but not execution times.

As test circuits, C1355 and C7552 circuits were used. Based on the experimental results, the predicted performance of each simulation protocol is analyzed. Gates were statically and randomly assigned to processors. Among active gates in a processor, an active gate with the smallest local simulation time is chosen as a final active gate for a simulation cycle. Figures 4.5, 4.6, and 4.7 show the comparisons of experimental results and predicted performance in synchronous simulation, conservative simulation, and optimistic simulation, respectively. In the graphs, the predicted performance is compared with the experimental results as the GP ratio increases.

In general, the parallelism obtained from the experiments is a little lower than the predicted performance. But, optimistic simulation has almost the same performance

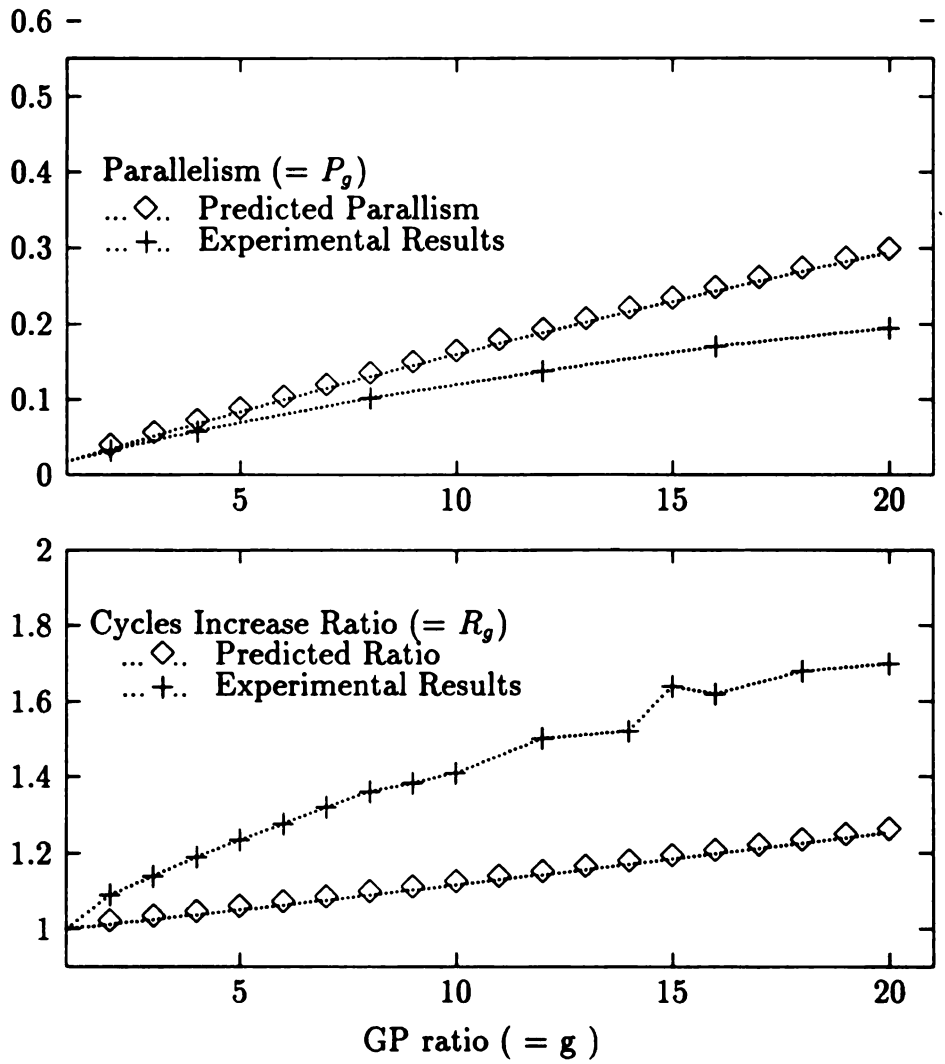


Figure 4.5: Comparisons for C7552 in synchronous simulation

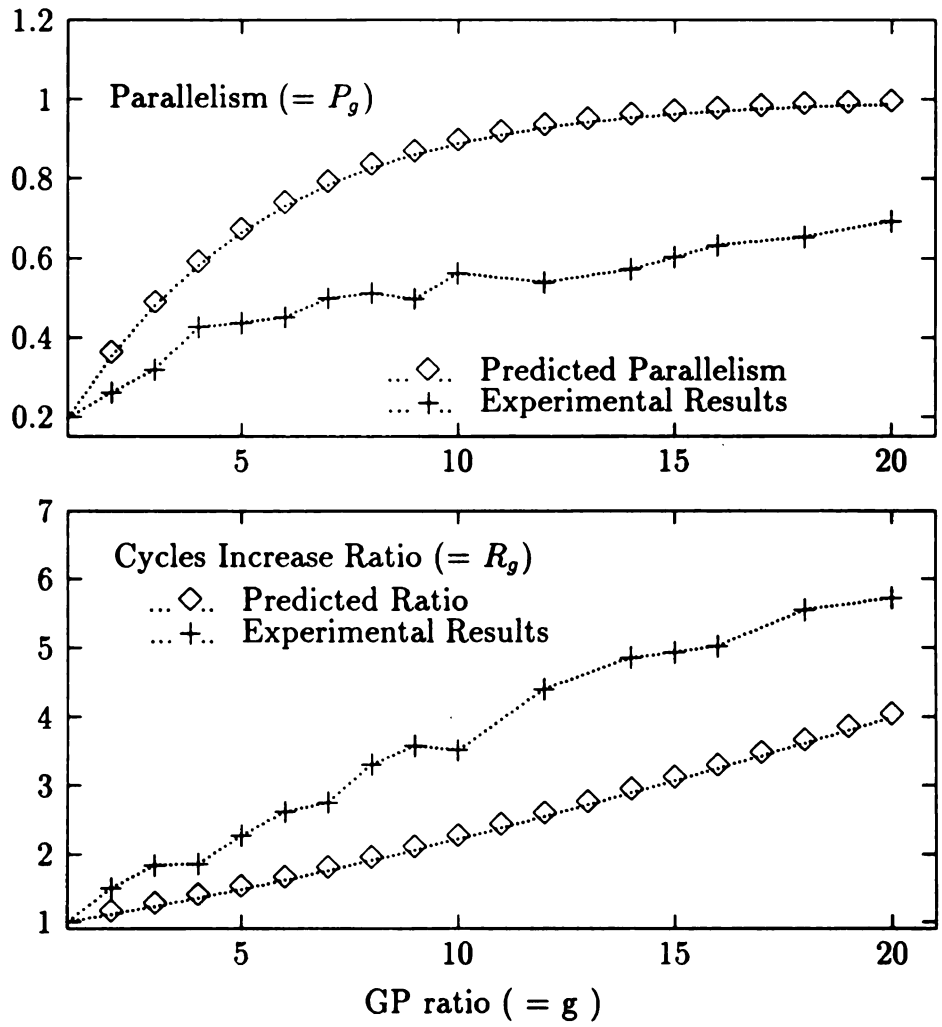


Figure 4.6: Comparisons for C1355 in conservative simulation

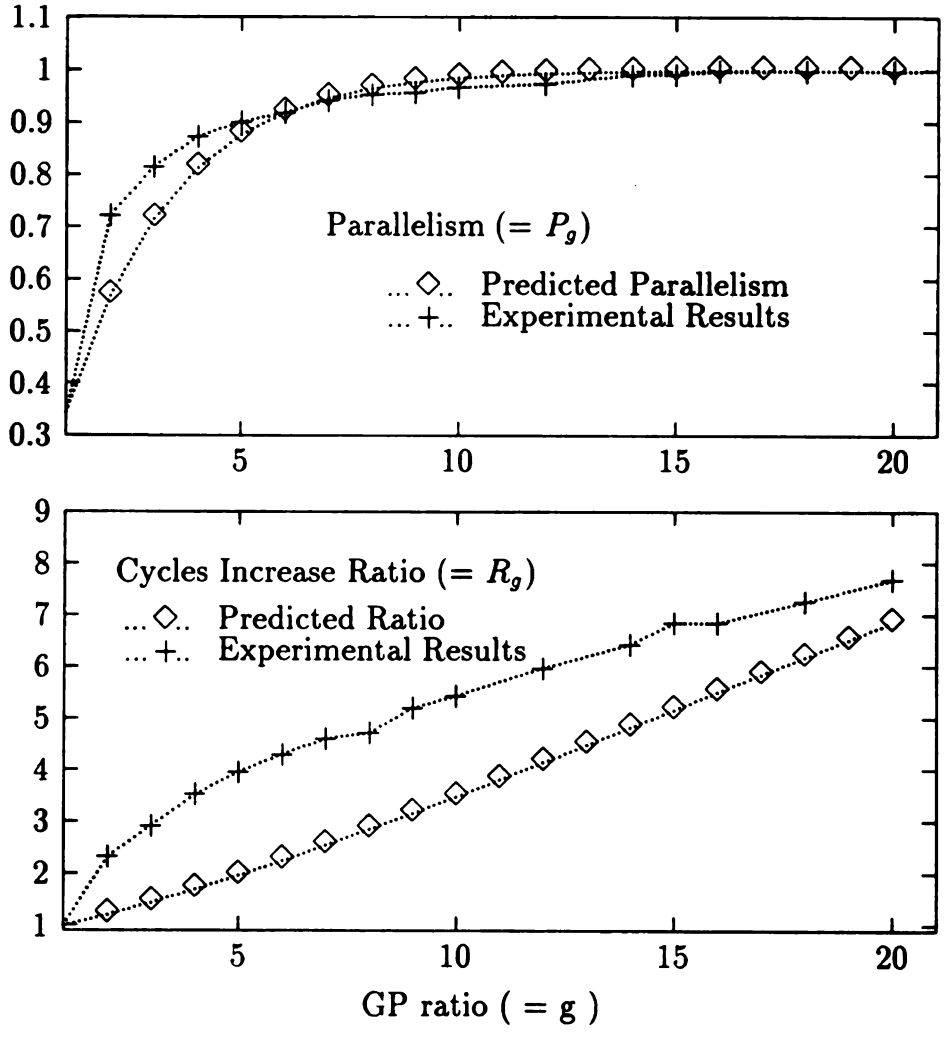


Figure 4.7: Comparison for C1355 in optimistic simulation

as measured by parallelism, as predicted. Parallelism depends on how gates are selected among active gates in a processor for execution. According to experience, the selection of a gate with the smallest simulation time gives much better performance than selection based on round-robin. The reason is that the former technique reduces the frequency of rollback in optimistic simulation and facilitates the update of link clocks in conservative simulation. Among the three simulation protocols, conservative simulation shows the biggest difference between the predicted parallelism and the observed performance. We can explain this observation as follows. When the GP ratio increases in the conservative simulation with null messages, link clocks are not updated as soon due to the following two reasons. First, the number of active gates which will not be chosen for execution during a simulation cycle increases as GP ratio increases. The gates cannot propagate events to update corresponding link clocks for the simulation cycle. Second, null messages cannot be propagated in a proper fashion in each simulation cycle because all except one gate are not allowed to send even null messages to successors. Thus, as GP ratio increases, the number of gates which cannot send any events or null messages increases.

Let us compare the predicted increase ratio of simulation cycles and the experimental results. According to the graphs, the observed increase ratios of simulation cycles are much higher than the prediction. In optimistic simulation, the predicted ratio and the observed ratio become close when GP ratio is high. Among the three simulation protocols, conservative simulation gives the biggest difference between the predicted ratio and the observed ratio. The reason for this is the same as for the difference in parallelism. The number of simulation cycles is reciprocal to parallelism.

4.5 Conclusions

There are no massively parallel SIMD machines in which a processor can contain many gates because currently available systems have local memory limit problems. We presented a model to compute parallelism and number of simulation cycles depending on the gate-to-processor ratio when we know the active ratio. Based on the model,

we can predict the performance of parallel logic simulation for a circuit with more gates than processors.

The predicted performance was compared with the experimental results for some benchmark circuits. The predicted curves in the above figures have the same shape as observed results. We can also predict execution times based on the predicted number of simulation cycles and time taken per simulation cycle.

Chapter 5

Enhancement of Simulation Clock Advancements

In parallel logic simulation using traditional distributed event-driven simulation protocols, such as the Chandy-Misra algorithm and Time Warp, an event is used to carry a value associated with a timestamp. Each gate computes a local virtual time (LVT) which is the smallest timestamp of unprocessed events. All the events with timestamps equal to LVT are involved in event evaluation at each gate during a simulation cycle (or iteration) based on the event selection (or scheduling) policy of a simulation protocol. Event evaluation means that an output value is computed based on the gate type with *all* the chosen events. After event evaluation, an output event may be propagated to successors.

We propose efficient event evaluation and propagation techniques to enhance simulation clocks of conservative simulation and optimistic simulation on parallel processing environments. The main idea of the techniques is to allow more than one event evaluation per simulation cycle and to pack more than one propagation event in a single message. In other words, evaluation of many events in each simulation cycle is allowed, reducing the number of simulation cycles, communication costs, and execution times. A sequence of events to be propagated as a single message is called a *multi-event* in this chapter. In parallel processing environments, communication overhead becomes very significant as the number of processors increases. With multi-events, we can significantly reduce communication cost since a set of events is sent as a single message.

In conservative logic simulation, each gate computes an advancement window containing events which can be safely executed without violating event execution precedence. In optimistic simulation, an aggressive advancement window is given

which contains events to be aggressively evaluated. More than one evaluation is allowed per simulation cycle even though rollback frequency increases.

The proposed advancement windows and multi-event techniques were implemented for some ISCAS'85 and '89 benchmark circuits on the CM-2. Good performance, measured by parallelism and execution time, was obtained for some benchmark circuits. We investigate the effects of window sizes on performance. As the window size increases, execution time decreases initially, but then increases since time taken for each simulation cycle increases due to the increased window size.

The remainder of this chapter is organized as follows. Section 5.1 explains how a set of events are sent as a single message. Both conservative simulation and optimistic simulation with advancement windows are presented in Section 5.2. Section 5.3 gives the performance of the proposed techniques in terms of number of simulation cycles, parallelism, maximum queue size, rollback frequency, and execution times. In addition, communication costs will be measured as a function of message length to see how much multi-events can improve communication costs.

5.1 Multiple Events in a Message

Gate-level logic simulation is different from other simulations, such as queueing network simulation, high level circuit simulations, etc., since simulation is performed based on fixed propagation routing, fixed delay, good lookahead capabilities, and non-preemption. The fixed propagation routing allows a gate to execute more than one event in a simulation cycle based on a specific simulation protocol: optimistic or conservative.

We discuss multi-events, in detail, to be contained in a single message. A message is an object which transfers information between gates. The information carried by the message represents a set of events as follows.

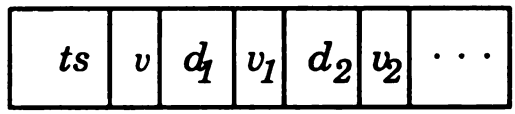
Definition 5 A *multi-event* contains a sequence of events to be propagated.

A multi-event contains a sequence of timestamps in non-decreasing order to propagate the results of the whole event evaluation within an advancement window. A

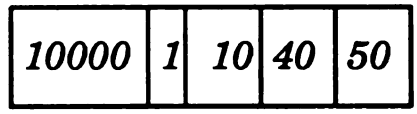
multi-event has the form as shown in Figure 5.1.(a). In the figure, *ts* represents the timestamp of the first event of a multi-event, and v indicates the signal value of the first event. In addition, d_i contains the timestamp difference between the first event and the $(i+1)$-th event in the multi-event, and corresponding signal v_i follows. In event-driven simulation, only events whose output signal is different from the output signal of the previous event evaluation must be propagated.

Depending on what parallel machine is used, the format of a multi-event may vary. For example, if the computation of the difference in timestamps takes too much time, the timestamp itself is sent instead of the difference. The signal fields (v or v_i) may not be necessary in a multi-event if binary signals, either 0 or 1, are used and we can make sure that the signal value of the last event in the most recently sent multi-event is different from the value of the first event in the multi-event to be sent.

Time Warp has an antimessage used to nullify previous incorrect event propagation. In this case, the message contains only a timestamp as its information.



a) the format of multi-event



b) an example of multi-event

Figure 5.1: A multi-event

For example, suppose binary signals are used and that there are propagation events e(10000,1), e(10010,0), e(10020,0), e(10030,0), e(10040,1), and e(10050,0). The message to be sent will contain the *multi-event* shown in Figure 5.1.(b). In traditional distributed event-driven simulation, at least 6 simulation cycles and 4 send operations

for the above event evaluation and propagation are required. With the proposed advancement windows and multi-events, however, all the above evaluations may be performed and their results will be propagated to successors during the same simulation cycle.

5.2 Clock Advancement Windows

In this section, the concepts of clock advancement windows in conservative simulation and optimistic simulation are discussed. Since the advancement window techniques are implemented on a massively parallel SIMD machine, terminology needed for implementation on the machine will also be introduced.

Definition 6 Event evaluation at simulation time t is defined to be *safe* only if it is certain that no event with timestamp less than t can arrive in the future. Otherwise, event evaluation is called *unsafe*.

5.2.1 Advanced Conservative Logic Simulation

Link clocks and minimum link clocks are used as defined in Chapter 2. That is, a link clock of each input port of a process is defined to be the timestamp of the most recently arrived event along the input link. The minimum link clock of a process is the minimum of link clocks of all its input ports.

Based on the concept of safety, we define the following advancement window in conservative simulation.

Definition 7 In conservative simulation, a window at a gate can be defined from LVT ($=l$) to minimum link clock ($= c$) where $l \leq c$. Such a window, which contains the events to be involved in safe event evaluation, is called a *Maximum Conservative Advancement Window* (MCAW).

On MIMD machines, all (or some) events within the maximum conservative advancement window can be executed for a simulation iteration (or cycle). All gates

are synchronized in SIMD processing environments. Hence, all gates must wait until the slowest (or busiest) gate finishes its event evaluations. To avoid that situation, we limit the number of event evaluations.

Definition 8 A *Synchronized Conservative Advancement Window* (SCAW) is defined as the synchronized window which contains only events to be involved in safe event evaluation. The size of the window is defined as the number of event evaluations from LVT.

Definition 9 If advancement windows are applied in logic simulation, each simulation cycle has a repeated procedure which consists of LVT computation, choosing active gates, and event evaluation. This procedure is called an *advancement cycle.* Only one event evaluation is allowed in an advancement cycle.

A simulation cycle has one or more advancement cycles. For example, Figure 5.2.(a) shows the state of a gate during simulation. The minimum link clock of the gate is 60. All events from LVT 10 to the minimum link clock can be involved in safe event evaluation, as shown in Figure 5.2.(b). In this case, an event evaluation means that an output value is computed based on all the events with the same timestamp. For example, the event evaluation at 10 in Figure 5.2.(b) involves two events stored in queues Q1 and Q2, respectively. All the events in the MCAW can be conservatively evaluated without violating event processing precedence. In other words, the MCAW consists of a sequence of safe event evaluations. In the figure, since SCAW size 3 is used, the maximum number of LVT computation and event evaluations at a simulation cycle is three.

Each gate might have a different MCAW at a certain simulation cycle, as shown in Figure 5.3. Gates 1, 2, and 3 have MCAW of sizes 5, 2, and 4, respectively. We assumed that SCAW size is 4. A gate whose MCAW is not smaller than a SCAW, as shown in gates 1 and 3 of the figure, performs event evaluation at each advancement cycle of the simulation cycle, as shown in gate 3 of the figure. Otherwise, the gate may be idle after the first few advancement cycles. It is very important to choose an appropriate SCAW size for good performance. The performance will be

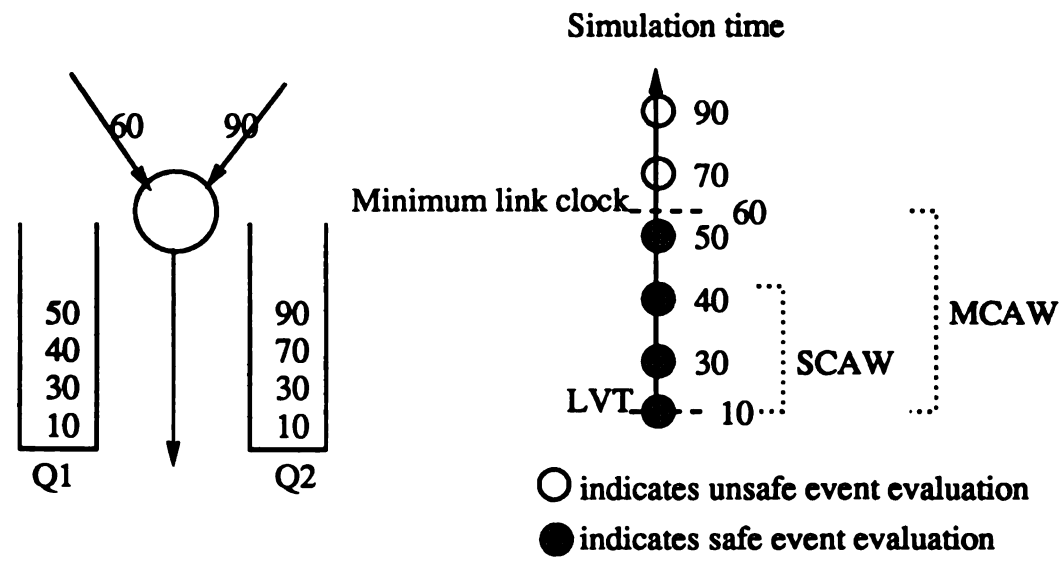


Figure 5.2: An example of an advancement window

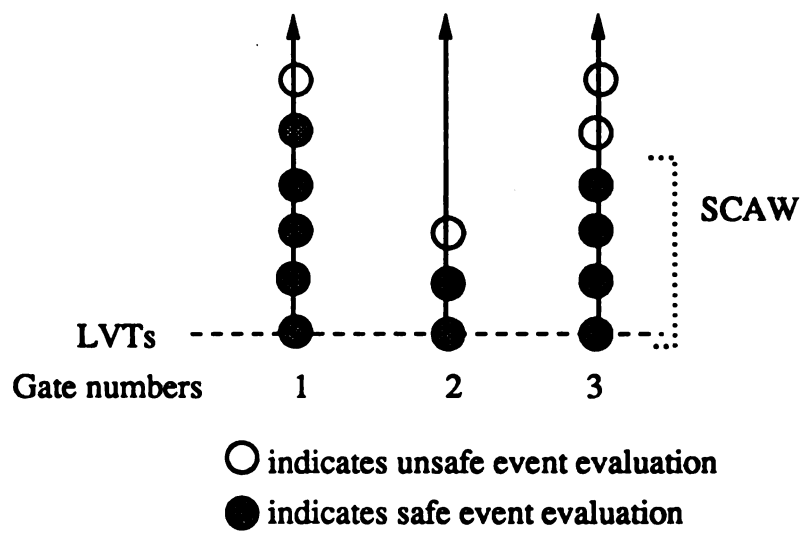


Figure 5.3: Relationship between MCAWs and a SCAW

analyzed according to the the SCAW size in Section 3. In advanced conservative logic simulation with advancement windows, the following steps are performed at each gate for each simulation cycle.

Algorithm ADVANCED CONSERVATIVE

1. The minimum link clock is computed.
2. The following advancement cycle is repeated as many times as the predetermined advancement window size.
   (a) LVT is computed.
   (b) if LVT is less than or equal to the minimum link clock, event evaluation is performed.
3. Event propagation is performed.
4. Each link clock is set to the last timestamp in the *multi-event* received on the link.
5. Queue manipulation is performed if necessary.

In addition, we can use a global clock to choose active gates for better performance if the above algorithm is implemented on SIMD machines, as given in [25]. For the above algorithm, a multi-event may also carry a virtual time which is the delay of the gate plus the last LVT. This value is useful in setting the link clocks of successors. Conservative simulation with advancement windows can be applied to the null message strategy as well as the deadlock detection and recovery strategy. It is known that conservative simulation with deadlock detection and recovery is very efficient on MIMD machines [20, 63], while the simulation with null messages works efficiently on massively parallel SIMD machines [25]. The techniques proposed in this section can be applied effectively in both cases.

5.2.2 Advanced Optimistic Logic Simulation

In traditional optimistic simulation such as Time Warp, only one LVT increment is allowed at each simulation cycle. In the proposed approach, each gate can advance

its LVT one or more times based on a predetermined advancement window size. We define a term for the window.

Definition 10 A *Synchronized Aggressive Advancement Window* (SAAW) is defined as the synchronized window which contains some events from LVT for each gate as far as there are available events to be executed. The size of the window is defined as the number of event evaluations.

If a MIMD machine is used as a target machine, a predetermined advancement window, which need not be synchronized, is applied to each gate during simulation. Event evaluation in optimistic simulation is allowed even when ordering could be violated. In the proposed technique, event evaluation is performed for all events within a SAAW even if rollback may occur later. In advanced optimistic logic simulation with advancement windows, the following steps are performed at each gate for each simulation cycle.

Algorithm ADVANCED OPTIMISTIC

1. The following advancement cycle is repeated as many times as the predetermined advancement window size.
   (a) LVT is computed.
   (b) For the first advancement cycle only, rollback is performed if the LVT is less than or equal to the previous LVT.
   (c) Event evaluation is performed.
2. Multi-event propagation is performed.
3. Queue manipulation and fossil collection are performed, if necessary.

Since the timestamps of events within a simulation cycle never decrease, only the first advancement of a simulation cycle at a gate can cause rollback. Figure 5.4 shows a simulation example with SAAW size 2. That is, all events within the window are involved in event evaluation at the same simulation cycle. Gates 1 and 2 are busy during the simulation cycle, while gate 3 is idle after one advancement cycle.

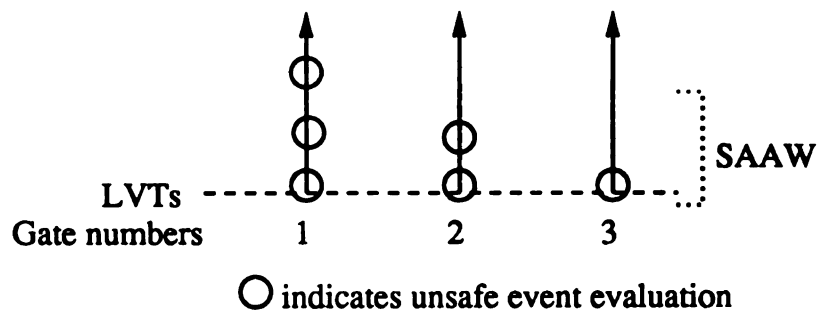


Figure 5.4: An example of SAAW with size 2

When the proposed approach is used, there are some advantages in queue manipulation. All events along the same link will be inserted near each other in the same queue. A disadvantage is that rollback frequency increases.

5.3 Performance Evaluation

Both conservative and optimistic logic simulations with advancement windows are implemented on the CM-2 with 32K processors. Both simulation protocols were implemented based on the same data structures proposed in Section 2. As test input, 1,000 randomly generated input vectors were used. In our simulation, we use the number of event evaluations as an advancement window size.

Let us discuss the performance metrics for the proposed techniques. The performance of the proposed simulation technique will be evaluated by means of the comparison to traditional distributed event-driven simulation techniques. We define the following performance metrics for the evaluation.

- Simulation Cycle Ratio (C_w), which is the ratio of the number of simulation cycles at window size w to the number of simulation cycles at window size 1. That is, simulation at window size 1 indicates the traditional distributed simulation techniques.

- Maximum Queue Ratio (Q_w), which is the ratio of the maximum queue size at window size w to maximum queue size at window size 1.

- There are two ways of defining parallelism based on how to define an active gate. When an *active* gate is defined as a gate which performs event evaluation at least once during a simulation cycle, *parallelism* is computed as

  $$\sum_{i=0}^{i=N} T_i/(NS)$$

  where T_i is the total number of simulation cycles for which gate i is active, N is the number of gates, and S is the number of simulation cycles.

  Parallelism Ratio (P_w) is defined as the ratio of parallelism at window size w to parallelism at window size 1.

  If we define an *active* gate as a gate which performs event evaluation at an advancement cycle, *adjusted parallelism* at window size w is computed as

  $$\sum_{i=0}^{i=N} T_i/(NSW)$$

  where T_i is the total number of advancement cycles for which gate i is active, and W is the synchronized advancement window size.

  When the window size is equal to 1, parallelism is the same as adjusted parallelism. The average number of clock advancements (or event evaluations) at window size w for a simulation cycle can be computed by multiplication of adjusted parallelism and the window size.

- Execution Time Ratio (E_w), which is the ratio of the execution time at window size w to execution time at window size 1.

- Rollback Frequency Ratio (R_w), which is the ratio of the rollback frequency at window size w to rollback frequency at window size 1 in optimistic logic simulation.

In terms of the ratios proposed above, we compare the performance of traditional event-driven simulation and proposed simulation techniques with advancement windows. The maximum number of event evaluations is used as a unit which determines advancement window size. In addition, there are several other ways to define the unit as follows.

- the maximum number of additional fields in a multi-event or
- the maximum number of events to be processed.

5.3.1 Advanced Conservative Logic Simulation with SCAW

The performance of advanced conservative logic simulation with different SCAW sizes is evaluated in terms of simulation cycle ratio, maximum queue size ratio, parallelism ratio, and execution time ratio. Figure 5.5 shows the performance comparison with traditional simulation for C1355, C1908, and C7552 with 1,000 randomly generated input vectors as a function of SCAW size. The performance in the figure was obtained based on the performance of traditional conservative logic simulation which evaluates gates once per simulation cycle.

As a measurement of how much local simulation clocks advance, the number of required simulation cycles is used. As the amount of clock advancement increases, the number of simulation cycles decreases, since local simulation clocks move ahead quickly. As shown in Figure 5.5.(a), as the SCAW size increases, the number of simulation cycles decreases rapidly and then converges to a certain point.

In Figure 5.5.(b), as the SCAW size increases, the maximum queue size decreases initially, but then increases. We can explain this as follows. Events in gates are properly processed and consumed as the SCAW size approaches a certain value in the circuit. In this case, the required maximum queue size would become small. But, as the SCAW size increases beyond this value, some gates may have large actually processed advancement windows, while other gates might have small windows. Therefore, some input ports might have a lot of events while other input ports have a

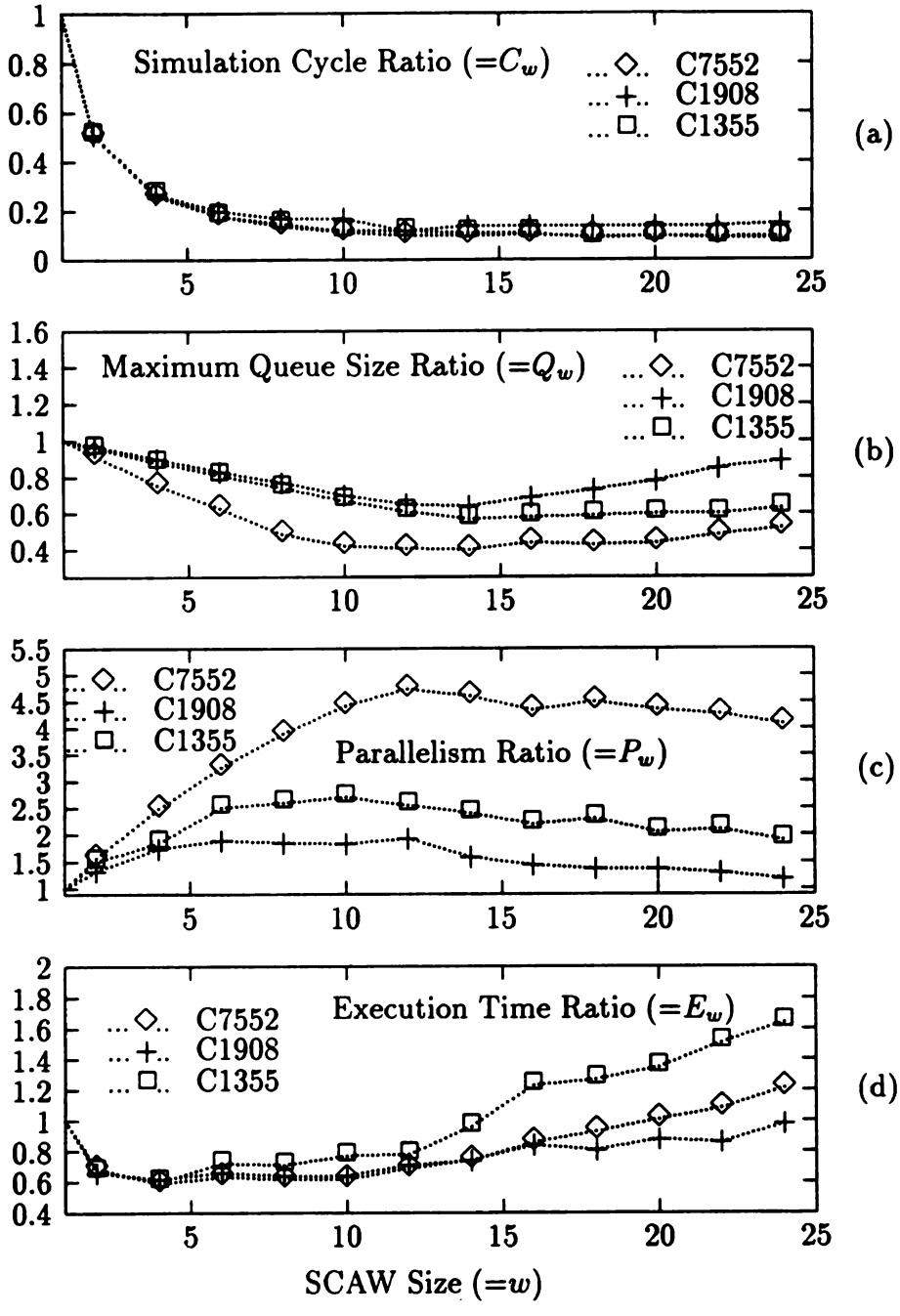


Figure 5.5: Conservative simulation with different SCAW sizes (1)

small number of events. There might be big differences between link clocks at a gate. Hence, there is a high possibility of having large maximum queue sizes.

In Figure 5.5.(c), the parallelism increases initially, but then decreases. We can explain this as follows. In the case of a large SCAW size, gates close to primary input ports process many events at each simulation cycle since the gates in general have large MCAW, while gates near primary output evaluate a relatively small number of events due to their small MCAW. That is, if a large SCAW is used, gates in the front part of the circuit complete simulation early, but gates in the rear part may have congestion problems. Therefore, as SCAW size increases beyond a certain point, parallelism might decrease.

In Figure 5.5.(d), the execution times decrease initially, then increase. The best performance is given when the SCAW size is 4. As the SCAW size increases, the time taken for a simulation cycle becomes longer since we need as many LVT computations and event evaluations as the SCAW size at each simulation cycle. Therefore, the best performance is given at the point where the time spent for a simulation cycle is not long, as well as the number of simulation cycles is not large.

Figure 5.6 shows the performance with respect to adjusted parallelism, and average number of advancements per simulation cycle. According to the graphs, as SCAW size increases, adjusted parallelism decreases since the number of idle advancement cycles increases. The average number of advancements increases, and converges to a certain value since there are no more useful advancement cycles if SCAW size exceeds MCAW sizes of all the gates in the circuit being simulated.

Figure 5.7 shows the performance of advanced conservative logic simulation as a function of the number of input vectors. SCAW size 4 is used since the best performance was given at that SCAW size. As the number of input vectors increases, the number of simulation cycles, the maximum queue sizes, parallelism, and execution times also increase.

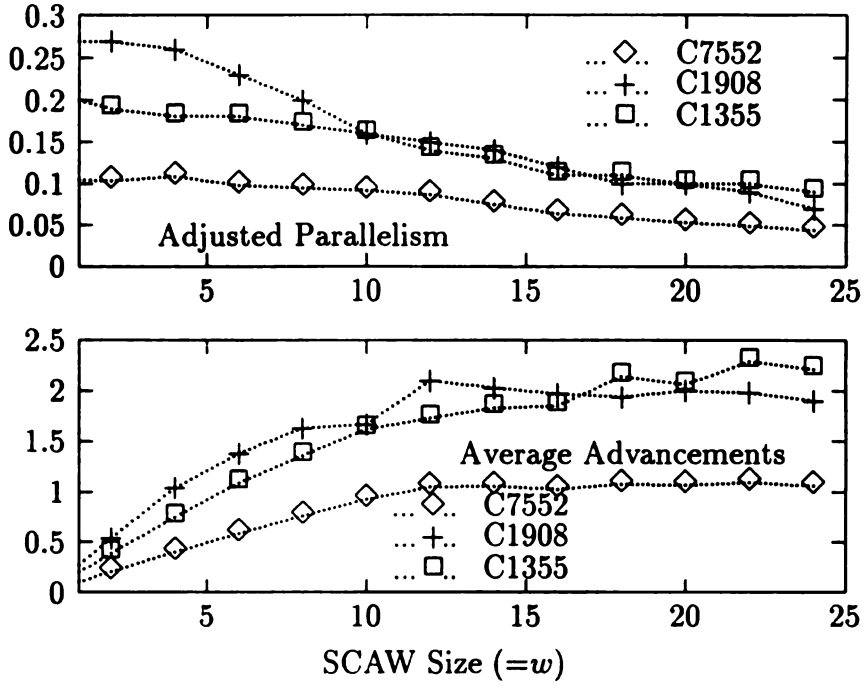


Figure 5.6: Conservative simulation with different SCAW sizes (2)

5.3.2 Advanced Optimistic Logic Simulation with SAAW

Figure 5.8 shows the performance for C1355, C1908, and C7552 as a function of SAAW size. In our simulation, the maximum number of event evaluations was fixed in advance as a synchronized aggressive advancement window. The performance comparison with traditional optimistic simulation was measured in terms of ratios for number of simulation cycles, parallelism, rollback frequency, and execution times.

As the SAAW size increases, the number of simulation cycles decreases and converges to a certain point, as shown in Figure 5.8.(a). The number cannot decrease further because rollback requires some number of additional simulation cycles even though clock advancements are enhanced by giving a larger window size.

In Figure 5.8.(b), as SAAW increases, rollback frequency rapidly increases. But, it decreases as the SAAW increases beyond a certain size. The reason is as follows. For a large SAAW, many events are processed in a simulation cycle, even though only the first advancement cycle of each simulation cycle may have rollback and the number of simulation cycles also decreases. At any rate, optimistic simulation with

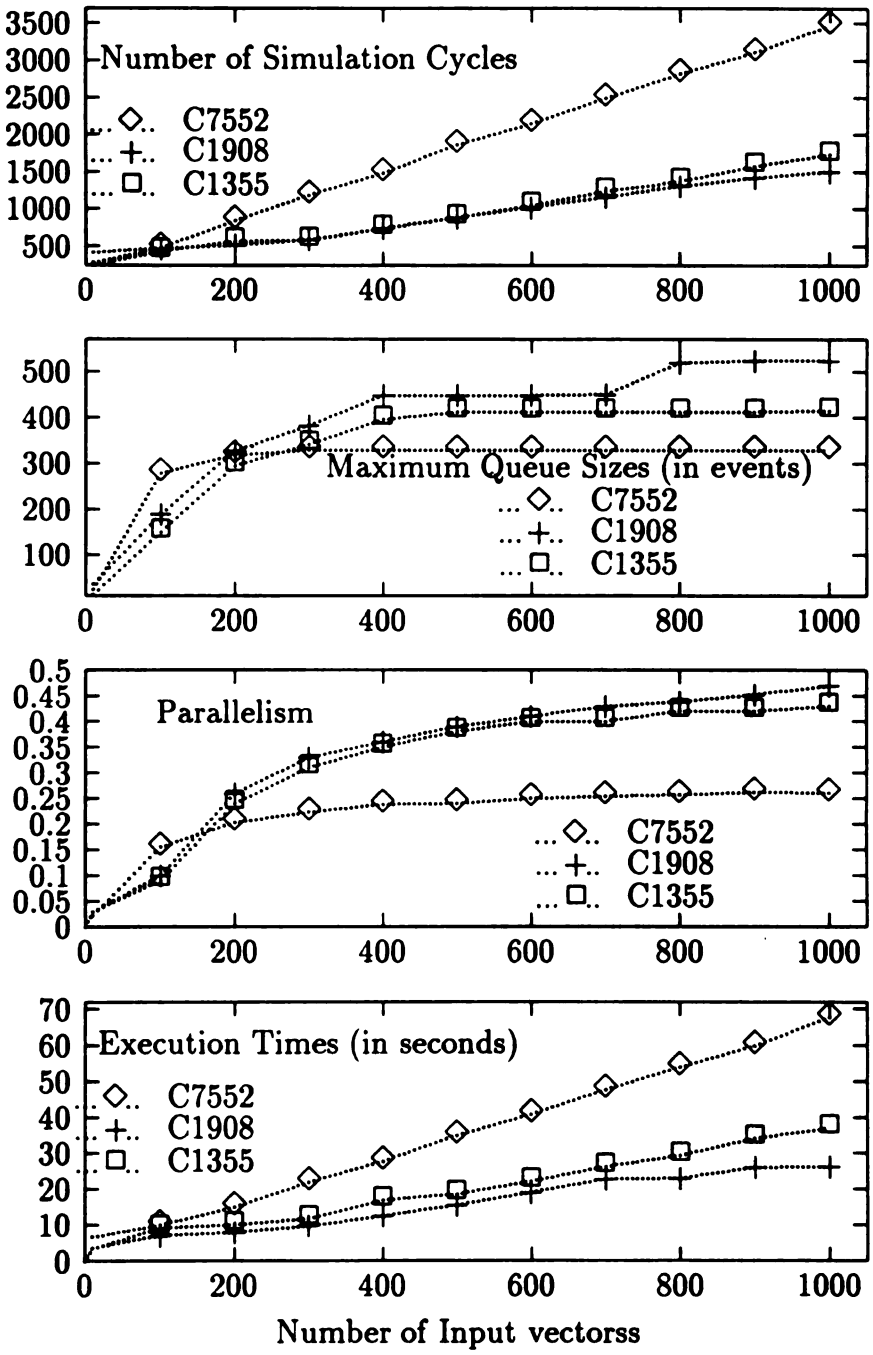


Figure 5.7: Conservative logic simulation with SCAW = 4

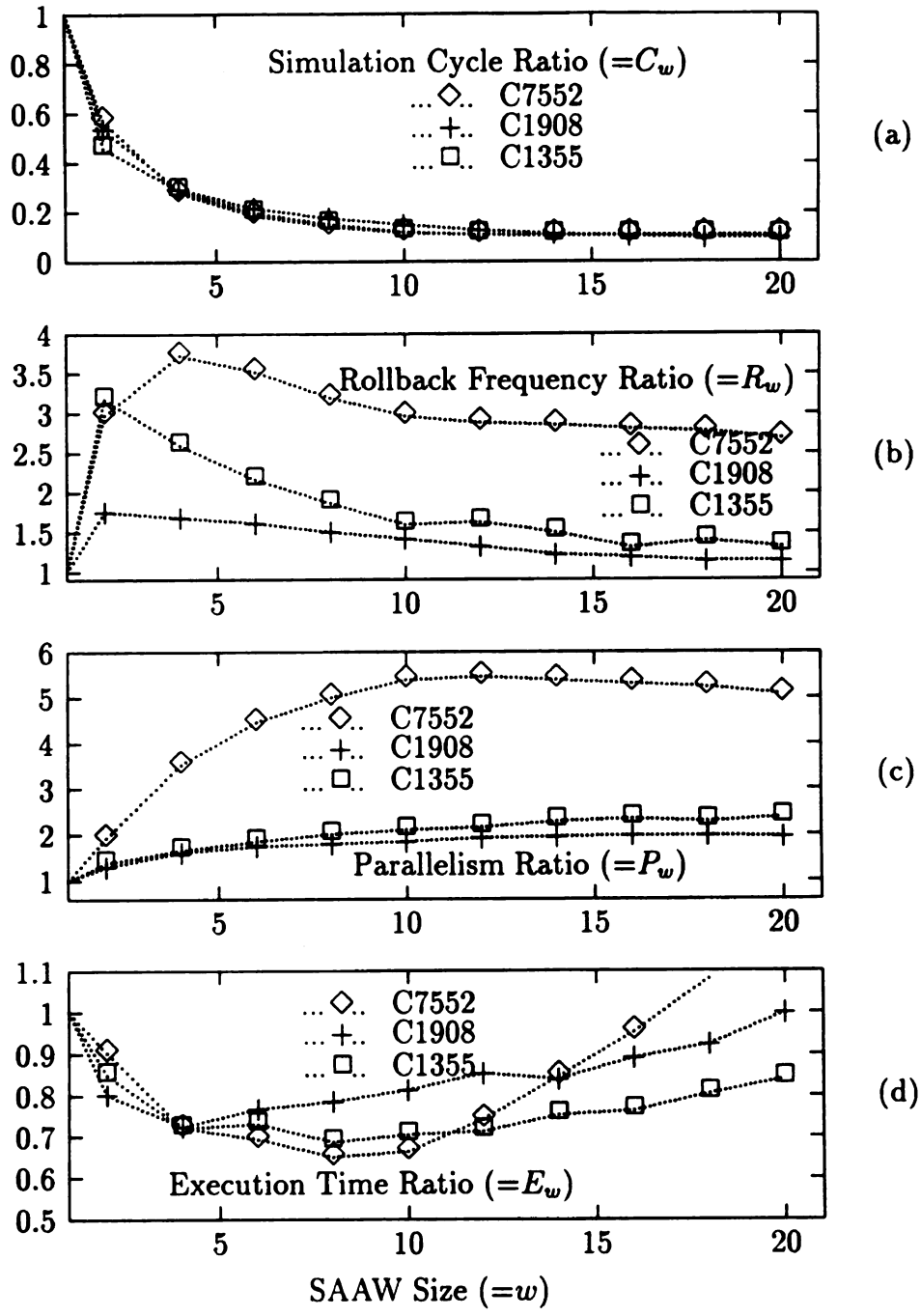


Figure 5.8: Advanced optimistic simulation with different SAAW (1)

advancement windows has much higher rollback frequency than traditional optimistic simulation.

As shown in Figure 5.8.(c), parallelism increases, since more events are involved in event evaluation at each simulation cycle as SAAW size increases. But parallelism decreases finally, since the number of active gates will become smaller for a large SAAW. With a large window size, the gate is more likely to consume its entire queues, and therefore be idle on the next cycle. For example, in the worst case, if the SAAW size is equal to the number of input vectors, parallelism will be extremely small.

In Figure 5.8.(d), the execution time of optimistic simulation with the proposed technique decreases initially for the following reasons. First, the number of simulation cycles decreases and only one rollback manipulation is required in a simulation cycle even though SAAW size increases. Second, queue manipulations are fast on circular binary search event queues. Finally, rollback manipulation is very simple, therefore high rollback frequency does not affect the execution time that much. But the execution time finally increases as the SAAW size increases, since the number simulation cycles converges to a certain point and each simulation cycle has more advancement cycles.

Figure 5.9 shows the performance measured in adjusted parallelism, and average number of advancements per simulation cycle with 1,000 randomly generated input vectors. Adjusted parallelism increases until SAAW size reaches 10, but the adjusted parallelism decreases since the number of idle advancement cycles increases beyond that window size. The average number of advancements per simulation cycle increases.

5.3.3 Performance on Circuits with Feedback

Fast simulation on circuits with feedback loops is one of the major issues in logic simulation. We measured the performance of conservative logic simulation with SCAW to see how the proposed window concepts affects the performance for sequential circuits. As test circuits, S9234, S13207, and S15850 are used which have D flip-flops and feedback loops.

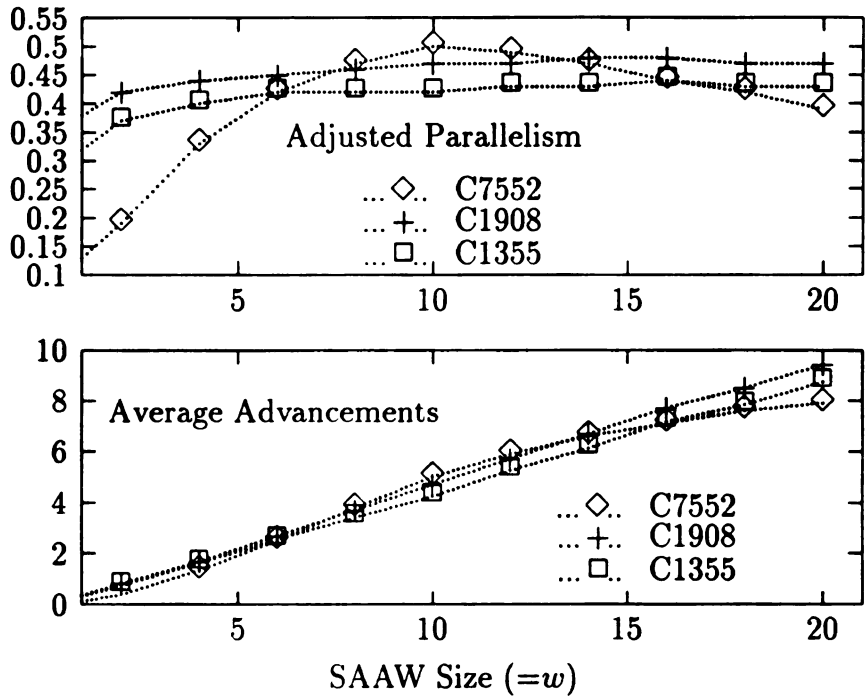


Figure 5.9: Advanced optimistic simulation with different SAAW (2)

Figure 5.10 shows the performance for the three test circuits as a function of SCAW size. According to the graphs in the figure, as the SCAW size increases, the number of simulation cycles decreases little. Parallelism decreases and execution times increase. There is no benefit to using the advancement window for circuits with feedback loops since advancement windows cannot work well for gates which have feedback inputs. The reason is that the gates with feedback inputs cannot advance their simulation clocks until they get input signals from feedback loops. How to efficiently use advancement windows on circuits with feedback is left for further study.

In addition, the performance of optimistic simulation with SAAW for sequential circuits was measured as shown in Figure 5.11. Depending on the benchmark circuits, the performance in the number of simulation cycles, parallelism, rollback frequency, and execution times, is different.

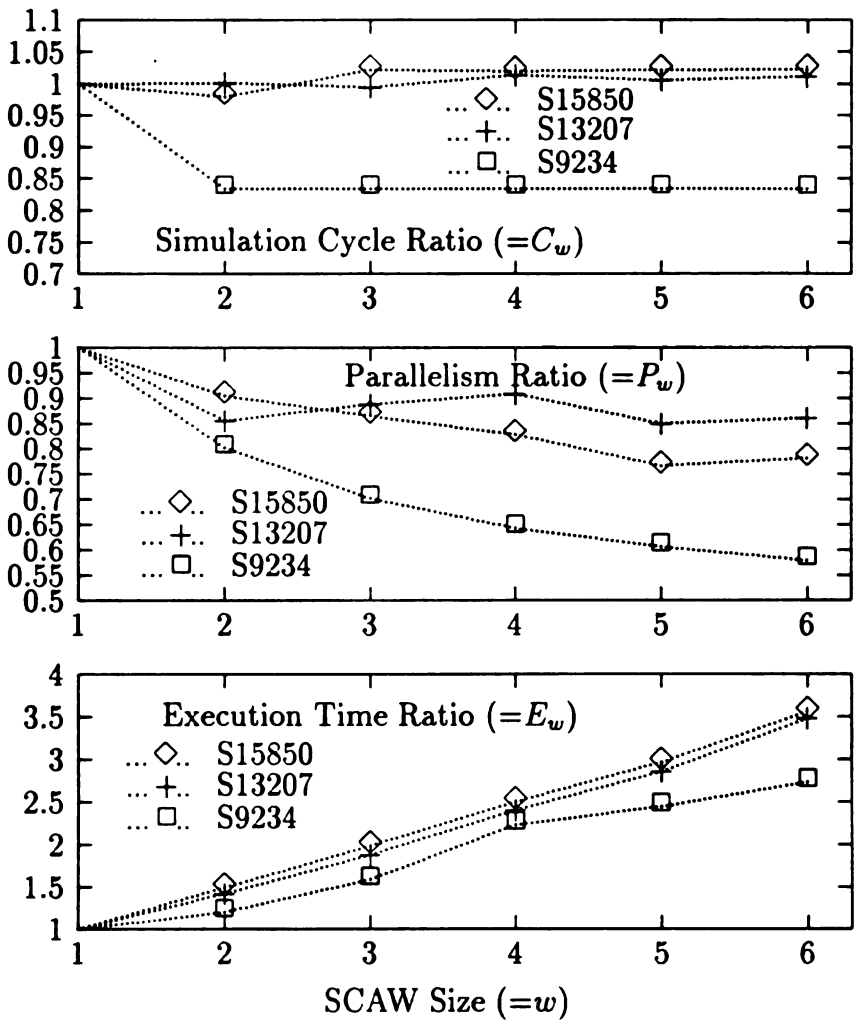


Figure 5.10: Advanced conservative simulation for sequential circuits

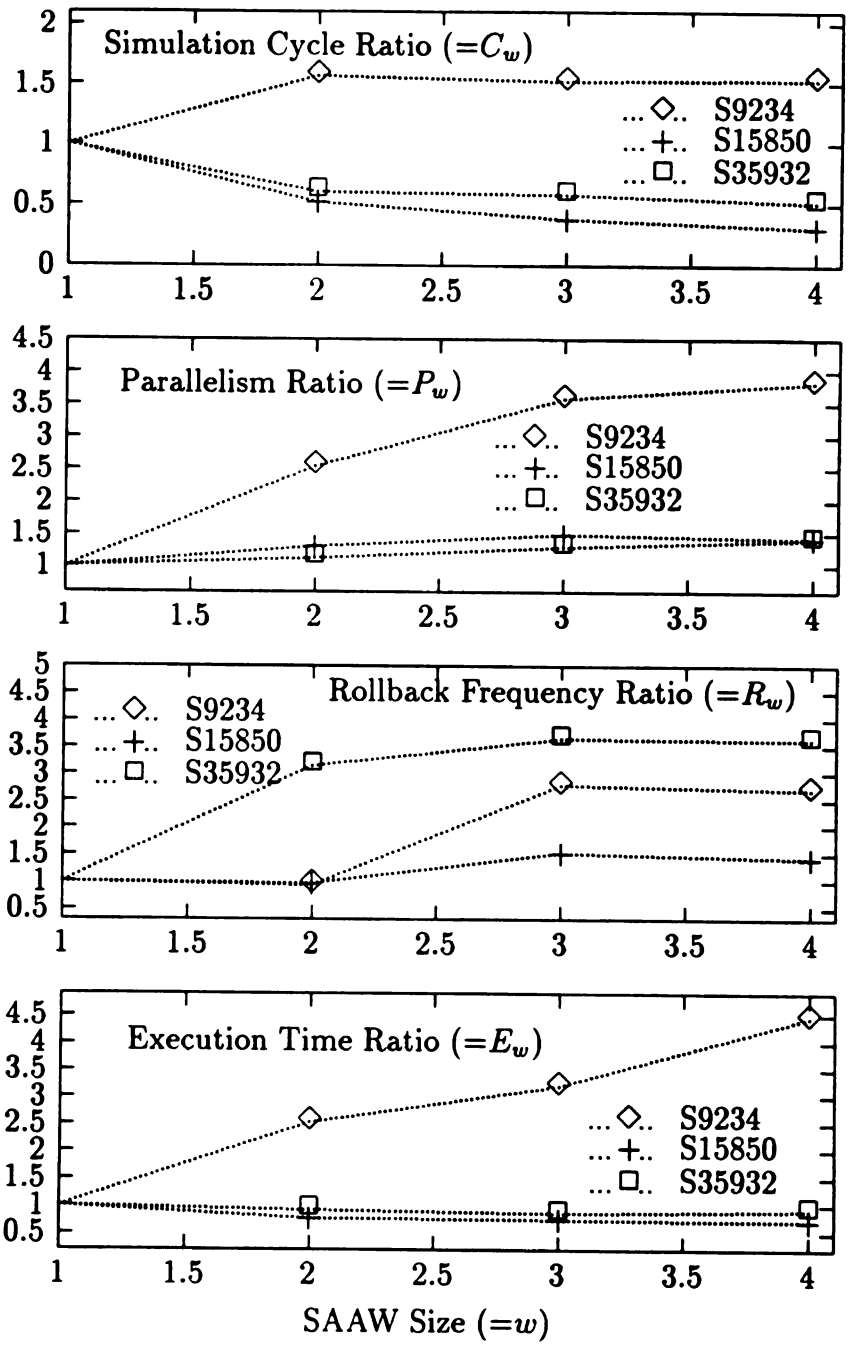


Figure 5.11: Advanced optimistic simulation for sequential circuits

5.3.4 Communication Costs According to Message Length

The performance of logic simulation using advancement windows will be different depending on the architecture of the parallel machine used for the simulation. Simulation with the windows can be done more efficiently in MIMD than SIMD environments. In SIMD machines, the longest time among all gates for event evaluations and multi-event propagation dominates a simulation cycle time, since all processors are synchronized. In MIMD machines, however, each processor can perform event evaluation and propagate multi-events at its own pace without affecting other processors.

The big advantage of using multi-events is to reduce the communication costs by grouping multiple events into a multi-event. In parallel processing environments, each message between processors contains some additional information, such as packet header and tail.

Figure 5.12 shows the *communication cost increase ratio* on several parallel machines as message lengths increase, where the cost increase ratio is defined as the ratio of communication time for a $4n$ byte message to communication time for a 4 byte message [9, 23, 44, 73]. In the figure, separate sending means that a single event is sent in a message, as in traditional distributed event-driven simulation. In this case, communication cost is exactly obtained by multiplying the number of events sent and time per event. In the figure, the CM-2 and the MP-1 are massively parallel SIMD machines and the others are MIMD machines.

We can achieve good performance in logic simulation with advancement windows on MIMD machines for the following reasons. First, according to Figure 5.12, communication costs on modern parallel machines such as the NCUBE-2 can be significantly saved by sending long messages rarely, rather than sending short massages frequently. Moreover, in SIMD processing environments, there might be many idle processes which have smaller advancement windows than the predetermined maximum advancement window due to synchronization of all processors. In contrast, in MIMD processing environments, there might not be many idle processes for smaller advancement windows.

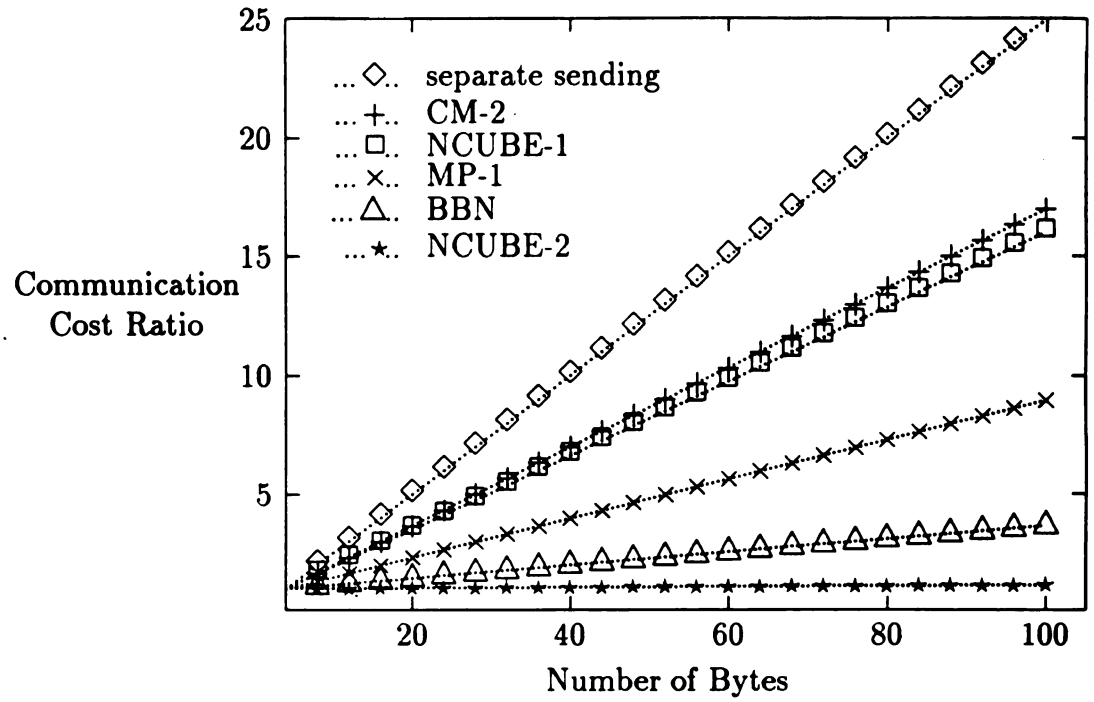


Figure 5.12: Communication costs according to message lengths

5.4 Conclusions

We have proposed new efficient logic simulation approaches with advancement windows to enhance local clock advancement for fast simulation in parallel processing environments. According to experimental results for some combinational benchmark circuits on the CM-2, both conservative and optimistic logic simulation protocols with advancement windows and multi-events require fewer simulation cycles and achieve higher parallelism than traditional conservative and optimistic simulation protocols. In addition, execution times also are smaller for some window sizes. Simulation with advancement windows is very promising when there are many input vectors, because average events per cycle will be almost at the maximum during steady state.

The good performance would be enhanced even more in MIMD environments than the SIMD environment of the CM-2. The technique can be applied to improve the performance of other simulation schemes, such as YADDES [74].

We can use the proposed technique to enhance local clock advancements, while a

global moving time window can be simultaneously applied to prevent queue overflow. That is, advancement windows make local clocks go ahead as fast as possible, but any local clock too far beyond a global clock is not allowed to advance to prevent queue overflow.

As future studies, we aim to implement conservative and optimistic simulation approaches with advancement windows and multi-events on MIMD machines and to compare with the performance obtained on a SIMD machine. In addition, we are going to continue research on how we can efficiently use advancement windows and multi-events for circuits with feedback loops.

Chapter 6

Distributed Token-Driven Logic Simulation

In this chapter, we present a distributed token-driven technique, which uses the concept of a dataflow network [68], for zero-delay logic simulation. Each signal is associated with a token. Tokens are used to synchronize all signals driven from the same input vector. A gate executes signals based on the readiness of tokens or comparison of token identifiers. Zero-delay logic simulation verifies functional correctness, not timing. The technique was implemented on the CM-2, and the BBN TC-2000, a shared memory MIMD machine. Experimental results indicate that the distributed token-driven technique outperforms both distributed event-driven simulation and compiled-code simulation for some benchmark digital circuits on the CM-2. A higher degree of parallelism can be achieved than with other compiled-code simulation techniques. With a reasonable partitioning scheme on the BBN TC-2000, the technique gives experimental results comparable to those obtained by a similar algorithm running on the CM-2.

The remainder of this chapter is organized as follows. Background and prior work for token-driven simulation is given in Section 6.1. Section 6.2 presents basic concepts of distributed token-driven logic simulation and explains how to efficiently apply the technique to parallel logic simulation on massively parallel SIMD machines. Section 6.3 presents performance evaluation for some benchmark circuits on the CM-2. In Section 6.4, the performance of token-driven simulation on a shared memory MIMD machine is presented.

6.1 Introduction

Due to the large amount of time required to simulate circuits at the gate level, much effort has been focused on finding more efficient simulation methodologies. Distributed event-driven simulation techniques have been studied for parallel logic simulation because they have the ability to easily handle asynchronous designs and timing analysis in parallel processing environments. Traditionally, digital logic simulation has been accomplished using the event-driven approach [14, 25, 35]. However, it has been shown that as much as 26 times more events are generated than necessary [70]. Recently, however, as the demands for maintainability, testability, portability, and high speed increase, synchronous circuits have been preferred for VLSI applications [69]. In an attempt to eliminate these extra events, techniques based on a concept known as compiled-code logic simulation have been proposed [16, 39, 69, 70].

Compiled-code logic simulation usually begins with a levelization process, which arranges the gates in levels, based on the minimum distance to the primary inputs. Once the levels are identified, then the logic at each level is compiled into executable code. The output vector for each input vector can be obtained by executing the code for each level sequentially from the first level to the last level. This method is faster than the event-driven strategy because there is no need to maintain a global event heap. LECSIM [70] is a simulator that incorporates both the compiled-code logic simulation approach and the event-driven approach. Another approach to logic simulation is the demand-driven approach [60]. In this methodology gates are only evaluated when their outputs are demanded. This eliminates many of the unnecessary events which were mentioned earlier. The main disadvantage of this approach is the overhead incurred by a recursive backtracking routine.

As discussed in [70], intermediate events in unit-delay simulation are used for detecting timing anomalies like hazards and race conditions. As the complexity of synchronous circuits increases, however, testing the functional behavior of a circuit is done before performing timing analysis. Zero-delay logic simulation is usually used for such functional testing. We are concerned with testing only the functional behavior

of the circuit.

6.2 Token-Driven Simulation

In conventional distributed event-driven simulation, each simulation output from a process is carried by an *event*, which is timestamped with simulation time at which it should be executed. The term *token* used in this chapter is conceptually different from an event. In distributed token-driven simulation, a token carries an output signal from a process without being timestamped. An identifier may be assigned to each token, if necessary, to synchronize messages for correct simulation.

In distributed event-driven simulation on a massively parallel SIMD machine, each process repeats the same procedure which consists of event selection, event evaluation, new event propagation and queue manipulation, as defined in Chapter 2. The time period to perform such a procedure is called a *simulation cycle* [25]. Each simulation cycle is synchronized in SIMD environments. In a similar way, the simulation cycle can be defined in the distributed token-driven simulation by substituting *event* with *token* in the above procedure.

Depending on the type of circuit to be simulated, the distributed token-driven technique for zero-delay logic simulation can be divided into two schemes: *without identifiers* and *with identifiers*. The scheme without token identifiers is the most efficient, but works only for combinational circuits, while the scheme with identifiers works for sequential circuits with feedback loops.

Let us consider the scheme without identifiers. An *active port* at a gate is defined to be a port which is supposed to receive tokens during simulation. Each active port has its own queue to store tokens. During every simulation cycle, the following steps will be performed:

Algorithm TOKEN-DRIVEN SIMULATION

1. For all gates, the gate is chosen as an active gate if each active port has at least one token.

2. **Each active gate produces an output signal from input signals of the tokens.**

3. **Each active gate propagates a token with the output signal to all the successors of the gate.**

4. **Each gate which receives new tokens puts them into its corresponding queues.**

Since simulation without identifiers is performed for a combinational circuit, the simulation terminates only if all the active ports in the simulation network have empty queues. There are no intermediate output signals produced from simulation. Hence, it is very easy to recognize the resulting vectors without output control.

The intuition behind the token-driven approach is quite simple [26]. Instead of evaluating gates based on the occurrence of a previous scheduled timestamped event, gate evaluation is triggered by the presence of at least one token in each of the gate's input queues. Although the token-driven approach and the event-driven approach appear to be quite similar, they are, in actuality, quite different. In the event-driven approach a gate is evaluated whenever an event occurs which references the gate. The occurrence of an event is based on its timestamp. In logic simulation, a single signal is usually associated with each event.

Tokens are used to synchronize all the messages from the same input vector. When a massively parallel SIMD machine is used as a target machine, the number of simulation cycles [25] required can be predicted and is smaller than any distributed event-driven simulation techniques.

Although a token may be augmented with an identifier to aid in signal synchronization, a timestamp is not needed. In the token-driven approach, gate evaluation occurs whenever a token is ready at every input port of the gate. By eliminating timestamps, we also eliminate the non-trivial problem of maintaining global time over multiple processes. This is a key factor in the performance improvements realized by the token-driven approach. An explicit timing mechanism can be ignored because only the functional behavior of the circuit is being tested. That is, given the input vectors, determine what the corresponding output vectors are. Timing anomalies such as hazards and race conditions are not considered. The token-driven

approach also differs from the levelized compiled-code techniques. In the levelized compiled-code techniques the circuit description is compiled directly into code. Input vectors are then evaluated by executing the code once for each vector.

6.2.1 Additional Considerations

There are many factors which affect the performance of the proposed token-driven simulation technique. The remainder of this section will discuss these factors.

Token Size

In most distributed event-driven simulation techniques, events, each with a digital signal, traverse a simulation network model for a circuit under simulation. On a parallel machine which has processors and slow communication, the communication overhead may cause significant degradation of the simulation performance. To cope with the overhead in the distributed token-driven simulation technique, digital signals are encapsulated into tokens. A token may contain one or more signal values. In this case, both the number of simulation cycles and the simulation time can be significantly reduced.

As the number of input vectors increases, good performance can be obtained if the token size increases accordingly. However, the performance goes down if the token size exceeds a certain limit since there is a trade-off between communication time reduction and evaluation delay. The optimal token size for the best performance depends on both the critical path length of a circuit and the number of input vectors.

Simultaneous Signal Evaluations

In distributed event-driven simulation on massively parallel machines, each process has a function table which contains each output signal for all possible input signal combinations according to the function of the process [25]. Only signals carried by events with the same timestamp can be evaluated at a process during each simulation cycle since only one signal is associated with an event. In other words, signals for at

most one input vector are involved with signal evaluation.

In the proposed distributed token-driven simulation technique, however, each active gate can simultaneously evaluate as many signals as carried by a token. In other words, signals produced from several input vectors can be simultaneously evaluated at a process because a token carries more than one signal and each output signal is independent of previous output signals.

A function table is modified for simultaneous signal evaluation in the distributed token-driven simulation technique. Let m and n be the number of input ports at a process and the token size in bits, respectively. Since each input port is associated with a circular queue to store tokens, a process needs m queues, Q_0, Q_1,... Q_{m-1}. Let t_{ij} be the j-th token in Q_i. In addition, b_{pqr} is defined as the r-th input signal in token t_{pq}. In this case, the first tokens t_{i0} $(i = 0..m-1)$ in each queue contain the following input signals:

t_{00} $(b_{000}, b_{001}, b_{002}, .., b_{00,n-1})$,

t_{10} $(b_{100}, b_{101}, b_{102}, .., b_{10,n-1})$,

.....

$t_{m-1,0}$ $(b_{m-1,00}, b_{m-1,01}, .., b_{m-1,0,n-1})$.

Here, let s be the number of input vectors which can be involved with simultaneous signal evaluation. In other words, all signals derived from s consecutive input vectors can be simultaneously evaluated at a process if each input port of the process has a token. The first s signals of the first tokens in each queue are involved with simultaneous signal evaluation using a function table lookup operation. A function table contains s output signals in each element. An index into the function table for the table lookup operation is defined to be the concatenation of the following bits:

$b_{m-1,0,s-1}, b_{m-2,0,s-1}, .., b_{00,s-1}$,

$b_{m-1,0,s-2}, b_{m-2,0,s-2}, .., b_{00,s-2}$,

.....

$b_{m-1,00}, b_{m-2,00}, .., b_{000}$.

We now discuss the data structure for a function table, and how each output signal of an element of the function table must be computed. The function table is a two-dimensional array. An element of the table is defined to be a binary value f_{ij}, (i=0,1,2,..,$(2^{(m*s)} - 1)$ and $j = s - 1, .., 1, 0$). For an index i of a function table, s elements (to be stored), $s_{i0}, s_{i1}, .., s_{i,s-1}$ are computed as follows:

1. index i is converted into a $(m * s)$-bit binary number, say

   $a_{m-1,s-1}, a_{m-2,s-1}, .., a_{0,s-1},$

   .....

   $a_{m-1,1}, a_{m-2,1}, .., a_{0,1},$

   $a_{m-1,0}, a_{m-2,0}, .., a_{0,0}.$

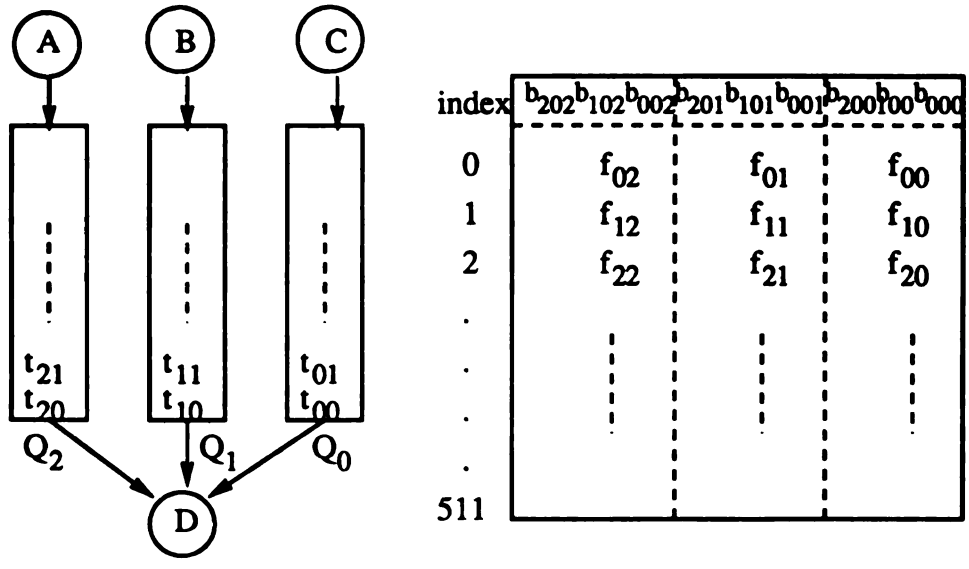
2. f_{ij} is computed by applying the gate function with bits $a_{m-1,j}, a_{m-2,j}, .., a_{0,j}$, where the gate function indicates a digital function of the gate like AND, OR, etc.

Given an index i in a table lookup operation, output signals f_{ij} $(j = s - 1, s - 2, ..., 1, 0)$ come out from the table.

Figure 6.1 shows the data structures for queues and a function table where $s = 3$ and $m = 3$ at a process. For best performance, the token size should be a multiple of s so that each signal evaluation is involved with s input vectors. The simultaneous signal evaluation strategy can be optionally used depending on the target machine. This strategy is recommended for parallel machines whose array access time is less than its functional access time.

6.2.2 Simulation on Circuits with Feedback Loops

The proposed distributed token-driven simulation technique can be applied to any circuit with feedback loops if each loop contains at least one flip-flop. The feedback loop can then be broken at the flip-flops by treating flip-flop inputs as primary outputs and their outputs as primary inputs, as is done in unit-delay compiled simulation [52].



a) Queues at process D

b) A function table at D

Figure 6.1: An example for simultaneous evaluation

In the distributed token-driven technique with identifiers, simulation ends when a predetermined identifier limit is reached.

A scheme using tokens with identifiers, which can simulate sequential circuits, is more general than the scheme without identifiers. During every simulation cycle for synchronous sequential circuits, the following steps will be performed:

Algorithm SIMULATION ON SEQUENTIAL CIRCUITS

1. For all gates, a gate is chosen as an active gate if each active port has at least one token.

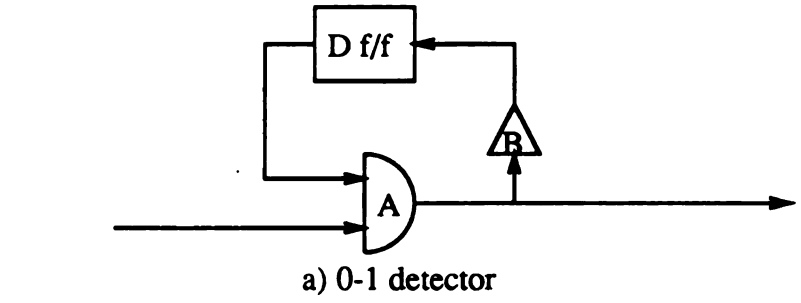
2. Each active gate produces an output signal from input signals of tokens with the same identifier.

3. Each active gate propagates a token with both the output signal and an identifier to all the successors of the gate.

4. Each gate which receives new tokens puts them into its corresponding queues.

In the second step of the above algorithm, all active ports must have tokens with

the same identifier. All flip-flops should generate a token with the same identifier as the first input vector at the beginning of simulation. Figure 6.2 shows a 0-1 detector with a D flip-flop, an AND gate, and an inverter, as a sample sequential circuit, which detects all the 0-1 sequences from input vectors. At the begining of simulation, the D flip-flop generates a token with identifier 0. For each input token to gate A from primary input vector, the D flip-flop generates the same identifier as that of the token. For example, for input token with identifier 1 and signal 0, the D flip-flop generates the token with identifier 1 from the previous input vector.



a) 0-1 detector

a) input vectors b) output vectors

Figure 6.2: Simulation on circuits with feedback loops

Before starting simulation, it is very important to determine how an identifier is assigned to a token. Numbers used as token identifiers must be non-decreasing. For example, timestamps of input vectors can be used as token identifiers. A flip-flop can be considered a gate in logic simulation. In step 3, the assignment of a new identifier is based on the function of the active gate. For example, a *normal* gate like AND or OR does not change the identifier, while a new identifier at a D flip-flop is set to the current identifier plus the clock interval. Asynchronous designs need more careful generation and assignment of identifiers than synchronous designs since asynchronous circuits are complicated.

The above procedure may not end since there might be some unprocessed tokens forever. To flush the unprocessed tokens at the end of the simulation, a token with infinity as its identifier is added to the end of input at each input gate. In this special case, a token with infinity as its identifier is considered to match any other token.

6.3 Performance Evaluation on the CM-2

The scheme for logic simulation proposed in [25] was used to implement the distributed token-driven simulation technique. The following data structures are considered. First, each gate in a circuit being simulated is assigned to a processor in the target machine. Second, each input port of a gate is associated with a circular FIFO queue to store tokens sent to the port since each port receives tokens in processing order. Each element of the queue has a link field to point to the next element. All queues belonging to a process must be contained in a processor. Finally, event evaluation is performed based on a table lookup operation as in [25].

The distributed token-driven simulation technique without identifiers was implemented on the CM-2 with 32K processors. A 32 $\times$ 32 array multiplier and ISCAS'85 and ISCAS'89 benchmark circuits [12, 13] are used as test circuits for the performance evaluation.

One of the advantages of distributed token-driven simulation is that the number of simulation cycles can be predicted. We can make the following observations.

Observation 1 Suppose that token size 1 is used. Let T and N be the critical path length (in nodes) of a test circuit and the number of input vectors for the simulation, respectively. The number of simulation cycles for the simulation is exactly

$$T + N - 1.$$

The above number can be obtained since no unnecessary messages (or events) are involved in simulation at all.

Observation 2 When a token contains more than one signal, the number of simulation cycles is computed as

$$T + \lceil \frac{N}{S} \rceil - 1$$

where S is a token size in signals.

Comparing experimental results of token-driven simulation to those in Chapter 3, token-driven logic simulation runs much faster than traditional event-driven simulation techniques. There are several reasons for the good performance. First, the number of simulation cycles is much less than in any other distributed event-driven simulation techniques. Second, communication costs are reduced since the size of a token is smaller than that of an event. In addition, grouping consecutive output signals into a token reduces communication costs. Third, queue manipulation for tokens is much simpler than for events. There are no unnecessary events during simulation. Finally, a simulation cycle is very simple.

Since there are no parallel zero-delay simulation schemes in existence, the experimental results of compiled-code simulation are considered for comparison. Table 6.1 shows the comparison of the simulation techniques for some ISCAS'85 benchmark circuits with 5,000 randomly generated input vectors. As a unit of measurement, *CM time* is used which is the time it takes to execute parallel instructions on the sequencer of the CM-2. In the measurement, only the time period for simulation is considered, not the time required for reading vectors, printing output, or translating the circuit description. LECSIM was measured on a SUN 3/260 [70]. The experimental results of both the PC-set and the parallel technique in [52] are shown in the table. The experimental results for token-driven simulation were obtained with token size of 21, i.e. each token contains 21 signals. According to the experimental results in the table, distributed token-driven simulation runs much faster than any other compiled logic simulation technique. Since the execution is pipelined, the execution times are proportional to the number of input vectors rather than the size of test circuits for a large number of input vectors. The execution time of token-driven simulation with up to 10,000 randomly generated input vectors is given in Table 6.2. In the measurements, a 32-bit array multiplier with 8256 gates and ISCAS'85/C6288 are considered as test circuits.

Test Circuits	LECSIM (SUN3/260)	PC-set	Parallel	Token-driven
C1355	309	84.9	9.8	2.8
C1908	500	162.7	54.3	3.0
C6288	1487	1757.3	369.3	4.6

Table 6.1: Performance comparisons (in seconds)

No. input vectors	2100	4200	8400	10000
C6288	3.59	4.99	7.77	8.84
Mul-32	3.38	4.70	7.16	8.10

Table 6.2: Execution times (in seconds)

We now compare the proposed distributed token-driven simulation with traditional distributed event-driven simulation. Output control is not required in distributed token-driven simulation since no unnecessary messages occur. Deadlock never occurs during simulation since each active gate unconditionally propagates a token to all the successors of the gate. Determining active gates based on the selective-trace approach does not cause any rollback and state-saving overhead as required in Time Warp. The computation of a global clock is not necessary because each gate works based on the readiness of tokens in its queues independently of processing of other gates. One of the great advantages of the distributed token-driven simulation technique is that queue manipulation is simple. In addition, the size of each queue is relatively small due to smaller token sizes compared with event sizes.

Depending on the token size, the execution time may vary since the size affects the number of simulation cycles, communication costs, and signal evaluation time. Figure 6.3 shows the performance as a function of token size. Token sizes 21, 53, and 117 are used for C7552 with up to 10,000 input vectors. The number of simulation cycles increases based on Observation 2. That is, token-driven simulation with larger token size requires smaller number of simulation cycles and larger communication cost

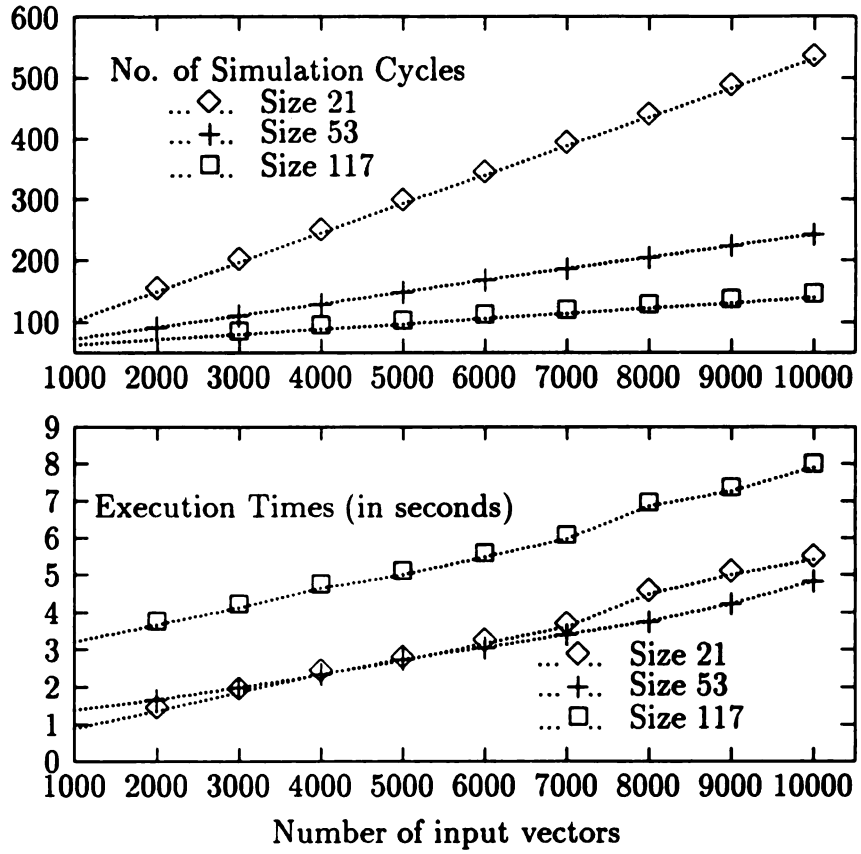


Figure 6.3: Performance according to token size

for each token. For a large number of input vectors, simulation with token size 53 has the best performance. Simulation with token size 21 requires more simulation cycles, while simulation with token size 117 has higher communication costs compared with other cases. In the CM-2, the most efficient token size is at least 21, since the optimal length of each element in an event queue for the best performance is a multiple of 32 bits and a part of an element is used as a link field in the queue.

We also measured the performance of token-driven simulation on sequential circuits with feedback loops. In the circuit with feedback, we cannot use the encapsulation of signals into a token. In other words, a token has a single signal and an identifier. In this case, the number of simulation cycles is much larger. Since event-driven simulation propagates events only when the output signal is different from the previous output, token-driven logic simulation has many more simulation cycles

Test Circuits	Simulation Cycle Ratio	Execution Time Ratio
S9234	2.39	2.1
S13207	2.00	1.77
S15850	2.86	2.55

Table 6.3: Performance comparisons to conservative logic simulation

than distributed event-driven simulation. Table 6.3 shows the performance comparison to conservative simulation with null messages. The second column contains the ratio of the number of simulation cycles for token-driven simulation to the number of simulation cycles for conservative logic simulation. The third column contains the corresponding ratio for execution time. According to the table, token-driven simulation runs much slower than conservative logic simulation for the given sequential circuits. This indicates that token-driven simulation has no advantages over event-driven simulation in circuits with feedback loops. However, token-driven simulation is easier to implement on a SIMD machine than event-driven simulation.

6.4 Experimentation on a Shared Memory Multiprocessor

Engelsma and we investigated the performance on a shared memory MIMD machine [31]. As was the case for the massively parallel SIMD algorithm, experimental results indicate that good performance can be obtained. Moreover, if we use a simple non-random partitioning scheme, performance comparable to that realized by the massively parallel SIMD algorithm can be obtained.

The token-driven simulation algorithm for MIMD machines is very similar to the algorithm used on SIMD machines. The only major difference is that, on MIMD machines, the processors will use a round-robin or time-slice mechanism to distribute simulation cycles among its set of assigned gates.

In order to obtain good performance with the token-driven approach in a shared

memory multiprocessor environment, data structures must be strategically distributed across the memories. For each gate, a simple data structure is allocated in the memory of the processor which has been assigned to the gate. This structure contains a gate type, pointers to the input queues of the gate's successor gates, and pointers to its own input queues. All gate input queues reside in the same memory that the gate structure resides in. A gate's output queues are another gate's input queues, which means that the output queue may be in a remote memory, or a local memory, depending on which processor the successor gate has been assigned to. Thus, the token queues can be thought of as mail boxes where processors can deposit tokens for another processor, or receive tokens from another processor. Each queue has a set of lock variables associated with them, which are used to ensure data consistency.

6.4.1 Performance Evaluation on the BBN TC-2000

The algorithm was implemented and run on a 44 node BBN TC-2000 Butterfly [31]. The circuits tested include 16x16 bit array multipliers, ISCAS'85/C1355, C1908, and C6288 [13], all of which are combinational logic circuits.

The most interesting results are the speedup curves which are illustrated in Figure 6.4. The speedup measurements are based on simulation runs on a 16x16 bit array multiplier using the consecutive gate partitioning scheme. In all cases except that of 1,000 inputs, considerable speedup is obtained by using between 1 and 8 processors. When 10,000 input vectors were used, the speedup curve is still relatively steep. This indicates that increasing the number of inputs will result in greater speedup. On the other hand, using less then 5,000 inputs resulted in inverse linear speedup after 8 processors. This is due to the overhead involved with generating the extra processes on the additional processors. Allowing for a longer steady state run time allows recovery from the initialization overhead. The maximum speedup attained for this circuit was approximately 8.5 by using 32 nodes and 10,000 input vectors. As the number of input vectors increases, we expect that the speedup can be improved as shown in Figure 6.4. From the figure, we can relate the Gustafson's scaled speedup [38]. In other words, the parallel portion of token-driven simulation is scaled up linearly with

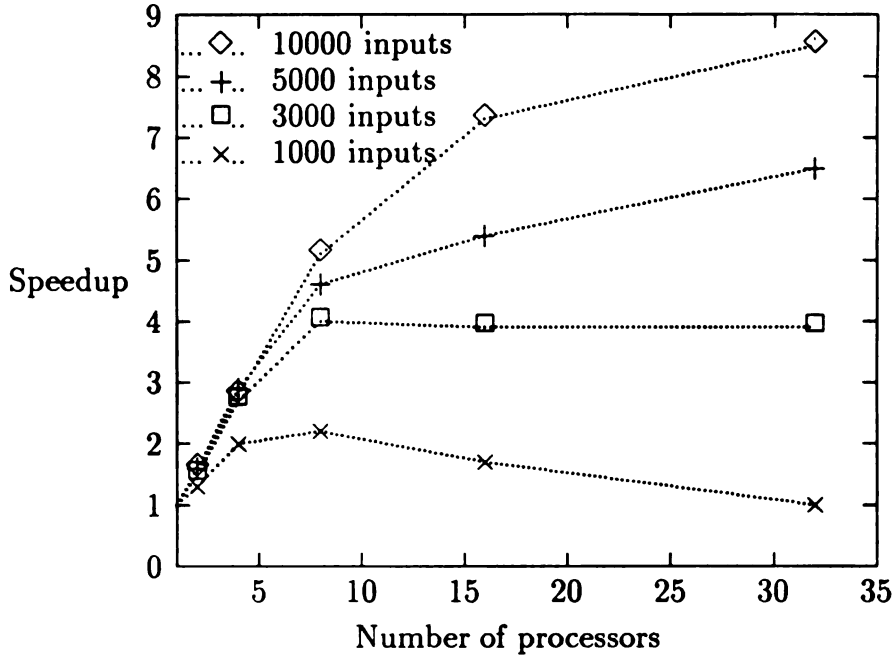


Figure 6.4: Speedup obtained for various numbers of input vectors

the number of input vectors, and the speedup increases linearly with the number of input vectors.

The simple consecutive gate partitioning scheme was implemented and compared to a random partitioning scheme. The experimental results indicate that by employing a very straight forward partitioning scheme, such as the simple consecutive gate scheme, performance is significantly improved. Figure 6.5 gives the speedup curves for 16 bit multipliers. For this circuit, the consecutive gate partitioning scheme reduced the execution time by approximately half. The maximum obtained speedups were 8.5 and 4.8 for 10,000 input vectors.

Representing signals as actual bits, and then using 31 bit logical operations to evaluate multiple sets of signals at a gate in parallel, resulted in significant performance increases. Figure 6.6 shows the difference in raw execution time of the bit packing approach compared to representing a single logical signal in an 8 bit byte. In general, speedup was not effected significantly. In addition to reducing the execution time, bit packing also allowed more input vectors to be processed, since less memory was required per signal.

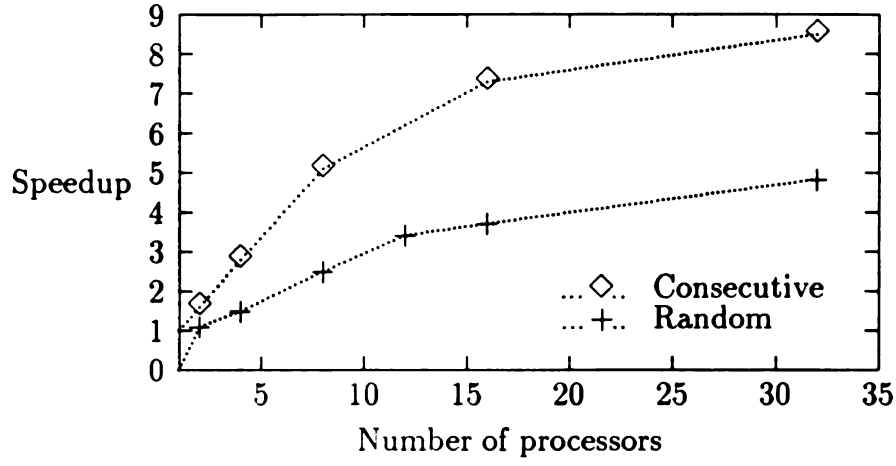


Figure 6.5: Speedup comparison for consecutive and random circuit partitioning

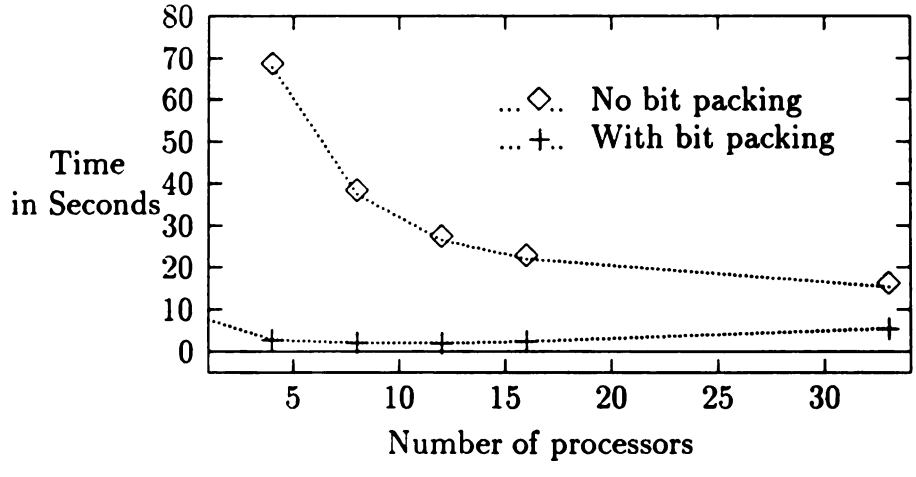


Figure 6.6: Bit packing comparisons

Circuits	BBN TC-2000	CM-2
8 bit multiplier	1.4	2.2
16 bit multiplier	4.3	2.7
C1355	2.9	2.9
C1908	3.6	3.1
C6288	6.2	5.3

Table 6.4: Performance comparison on MIMD and SIMD machines (in seconds)

The experimental results show that the MIMD implementation of the token-driven simulation approach delivers performance comparable to the SIMD implementation. The BBN TC-2000 and the CM-2 are considered as target systems for the MIMD and SIMD implementations, respectively. The BBN TC-2000 can support up to 256 processors, while the CM-2 supports up to 64K processing elements. It must be noted that, in these experiments, the TC-2000 used had only 44 processors. Table 6.4 lists the execution times (excluding input/output) of the various test circuits with 6,000 input vectors for both the MIMD and the SIMD implementations. With bit packing, each token carries 31 bits and 21 bits for the MIMD and SIMD implementations, respectively. Currently we can simulate the circuits with only 6,000 randomly generated input vectors due to the limit of total memory size.

Table 6.4 lists the execution times of the various test circuits with 6,000 input vectors for both the MIMD and the SIMD implementations. In the measurement, the execution time includes only the time period taken for simulation, but excludes time for reading vectors, printing output, or translating the circuit description.

The performance of token-driven simulation in MIMD environments depends on both the length of critical path [10] and the number of gates. On the CM-2, the number of processing elements is large enough to allocate one processing element for each gate for the benchmark tests. As the number of gates increases, the execution time on the CM-2 does not increase as much as on the BBN. On the BBN, however, the execution time increases since the number of available processing elements is much smaller than the CM-2.

6.5 Conclusions

The distributed token-driven simulation presented herein is a new efficient parallel zero-delay logic simulation technique in massively parallel processing environments, which integrates the advantages of distributed event-driven simulation and compiled-code simulation. The experimental results demonstrate that the presented technique in this chapter outperforms both traditional distributed event-driven simulation and compiled-code simulation when we consider only functional simulation rather than timing simulation.

The token-driven technique for zero-delay logic simulation has the following advantages: (1) high parallelism is achieved, (2) compared with distributed event-driven simulation, output control as well as unnecessary intermediate messages are not needed, and (3) a levelization procedure is not required.

A token-driven approach for parallel logic simulation on shared-memory multiprocessor machines has been presented. Using a simple partitioning scheme, experimental results indicate that significant speedup can be obtained for large circuits with a large number of inputs. According to the experimental results given in the above tables, distributed token-driven logic simulation is much faster, in both SIMD and MIMD environments, than any other compiled-code logic simulation techniques [52, 70].

As future work, we aim to investigate the optimal token size for a given critical path and the number of input vectors, and measure the performance of the distributed token-driven simulation technique with identifiers for sequential circuits with feedback loops based on the technique proposed in [26]. The optimal token size may vary for different circuits, possibly depending on the circuit topology. Further studies will consider the extension of the proposed distributed token-driven simulation technique to circuit designs with multi-delay.

Chapter 7

Conclusions and Future Research

In this thesis, we studied the performance of parallel logic simulation in massively parallel SIMD processing environments. The main objective was to model and improve the performance of gate-level logic simulation on SIMD environments, using distributed discrete event-driven simulation or complied-code simulation. We compared the performance of logic simulation based on parallel processing architectures, such as SIMD and MIMD. We found that massively parallel SIMD machines have many advantages over MIMD machines for logic simulation.

Our studies of parallel logic simulation in massively parallel SIMD environments involved

- experimental studies to evaluate the performance of parallel logic simulation using distributed event-driven simulation. The performance was evaluated with respect to number of simulation cycles, parallelism, maximum queue size, and execution times. Our experimental results provided an understanding of the performance of parallel logic simulation

- experimental analysis to evaluate the effect of clock advancement on performance of existing event-driven logic simulation protocols and ways to enhance clock advancements for better performance.

- probabilistic analysis to model and estimate the performance of parallel logic simulation as the ratio of the number of gates to the number of processors increases. The estimation was done in terms of number of simulation cycles, parallelism, and execution times. The estimated performance was compared with experimental results for three distributed logic simulation protocols, synchronous simulation, conservative simulation, and optimistic simulation.

- parallelization of a sequential simulation technique for fast zero-delay or unit-delay simulation to improve its performance. Our studies provided an efficient parallel version of compiled-code simulation.

We summarize the main contribution of our thesis now.

7.1 Summary and Major Contributions

This thesis presented broad studies of parallel logic simulation on massively parallel SIMD processing environments. We regard the followings as our contributions:

1. So far, most parallel logic simulation using discrete event-driven approaches have been implemented on MIMD machines. Three major distributed simulation protocols, synchronous simulation, conservative simulation, and optimistic simulation, have received the most attention. We examined how logic simulation can be done efficiently in massively parallel SIMD processing environments. Performance evaluations were given in terms of the number of simulation cycles, maximum queue size, parallelism, and execution times. We investigated efficient queue structures in each simulation protocol and proposed a new cancellation technique for optimistic simulation which is good for SIMD processing environments.

2. We found out that, despite theoretical arguments, optimistic simulation is not necessarily the best technique on massively parallel SIMD machines because of its inherent rollback and queue management overhead. Experimental results showed that optimistic simulation is much slower than conservative logic simulation, even though the number of simulation cycles is slightly smaller than conservative simulation. The reason is that optimistic simulation requires more time per simulation cycle. In addition, we observed that, in contrast to MIMD environments, conservative simulation with null messages is very fast on massively parallel SIMD machines.

3. We compared and analyzed the performance of massively parallel SIMD machine architectures for parallel logic simulation. The CM-2 and the MP-1 were target systems for the comparison. Experimental analysis was performed in terms of related machine instructions, local memory sizes, communication schemes, and

virtuality. The experimental results showed that the MP-1 is two to three times faster than the CM-2.

4. To reduce the rollback frequency and space overhead, the moving time window concept was applied to optimistic simulation. Use of this technique is necessary to prevent queue overflow on massively parallel SIMD machines. We found out that we can also use the technique in conservative simulation to avoid queue overflow.

5. We considered the assignment of multiple gates to a processor to increase both parallelism and system utilization. This is necessary since the number of physical processors is limited even though the circuits to be simulated may have a huge number of gates. On the current massively parallel SIMD machines, such as the CM-2 and the MP-1, only one gate is assigned to a processor since local memory size is small. According to announcements from supercomputer manufacturing companies, massively parallel SIMD machines with a large number of processors and large local memories will be available in the near future. In this thesis, we have proposed a statistical model to estimate the performance of parallel logic simulation when a processor of a massively parallel SIMD machine contains more than one gate. The performance was predicted in terms of the number of simulation cycles, parallelism, and execution times. We showed how to estimate the performance when the performance with one gate per processor is given. The predicted performance was compared with simulation results. The comparison showed that the performance estimation agrees with the simulation results for some simulation protocols. We contributed to the prediction of performance for a large circuit on future large massively parallel SIMD machines.

6. In traditional distributed event-driven simulation, processes communicate by sending events. An event is timestamped with a virtual time and a digital signal. Only one local clock advancement is allowed. In this thesis, we proposed new event evaluation and propagation techniques in both conservative and optimistic simulations to improve (or facilitate) clock advancements and reduce communication costs. In both techniques, an advancement window is first determined and then all events within the window are executed and propagated. Experimental results showed that the number of simulation cycles can be significantly reduced and that the techniques

also works well on MIMD processing environments. The technique can be applied to improve the performance of other simulation schemes, such as YADDES.

7. So far, only compiled-code simulation with zero-delay or unit-delay has been used to check the functional correctness of a digital circuit in VLSI design. Compiled-code simulation works only in sequential processing environments. Distributed token-driven simulation has been proposed for fast zero-delay logic simulation on parallel processing environments. Experimental results showed the proposed distributed token-driven simulation on massively parallel SIMD machines works much faster than compiled-code simulation. Analysis also showed that the distributed token-driven technique works well on MIMD machines if appropriate circuit partitioning schemes are used.

7.2 Future Research

In this section, we present some topics for future studies based on the research given in this thesis.

Strategies for Better Performance

We can consider the following strategies for better performance on a massively parallel SIMD machine. First, to reduce both required memory size and event queue manipulation time, we can investigate and compare efficient data structures for logic simulations on a specific machine, such as the CM-2 and the MP-1, based on its architecture. Second, to cope with the congestion problem on the MP-1, assignment of logic gates to processors is an important factor in optimizing performance. Finally, partitioning a circuit is also important for the efficient use of parallel virtuality.

Combination of Both Conservative and Optimistic Simulation Protocols

Conservative simulation allows each process (gate) to execute events as long as rollback is avoided, while optimistic simulation allows each process to evaluate events

even if rollback may occur. The above techniques each have their own inherent drawbacks, which are mentioned in Chapter 1. As future research, hybrid techniques could be developed. A process may be allowed to do either conservative or optimistic simulation.

We can consider two kinds of decision procedures for how a process selects one of the two simulation techniques. First, before simulation starts, one of the simulation techniques is selected for each process based on its characteristics, such as critical path length from input gates. Alternatively, a process may obtain the information for the decision during simulation.

In addition, instead of using a global moving time window for all active gates, we may apply local windows to each gate based on the information obtained during simulation.

Logic Simulation on Feedback Loops

Efficient simulation of feedback loops is one of the major issues in VLSI design. We consider use of the proposed simulation techniques on feedback loops. Of course, token-driven logic simulation and event-driven simulation with advancement windows also should be considered for the efficient simulation of feedback loops.

Extended Simulation Results

We used the CM-2 and the MP-1 as our target systems in this thesis. According to the manufacturer of the CM-2, Thinking Machines Corporation, CM-5 will be released in the near future. We have heard that the CM-5 is much more powerful than the CM-2. We hope to obtain more extended experimental results on the new machine to compare with results obtained on the CM-2, to analyze the effects of system architecture and performance.

We have claimed that our proposed simulation techniques, both conservative logic simulation with conservative advancement window and optimistic simulation with aggressive advancement window, will give give good performance in MIMD processing environments. We hope to analyze the performance of both proposed techniques on

MIMD machines, such as the BBN TC-2000 and the NCUBE-2.

Extension to Other Levels of Simulation

We can consider several levels of simulation in VLSI design. It could be interesting to study how to extend the parallel simulation techniques proposed for gate-level logic simulation to simulations in other levels, such as circuit level simulation and behavioral simulation.

Generalized Performance Model for Logic Simulation in SIMD

Several performance models of parallel logic simulation as a synchronized iterative algorithm have been proposed and compared with experimental results on MIMD machines. As future research, a generalized performance model of parallel logic simulation in an SIMD environment could be developed. We found out that, among three distributed event-driven logic simulation techniques, optimistic logic simulation has the smallest number of simulation cycles and the highest parallelism , while synchronous simulation has the largest number of simulation cycles and lowest parallelism. We did not propose any performance model to explain the simulation results.

More Accurate Performance Model for the Gate-to-Processor Ratio

We assumed steady state conditions in the model to predict the performance of logic simulation as the gate-to-processor ratio increases. To get more accurate performance predictions, the exact active ratio of gates should be statistically computed for different gate-to-processor ratios. Gates as well as processors were assumed to work independently in the model. We could find some relationship for accurate modeling. Refinement of our model remains as future research.

In the model, we assumed that the time taken for a simulation cycle is independent of GP ratio. But, for accurate estimation, we need to consider some factors which affect the simulation cycle time. For example, depending on the machines used, the number of required send operations might be different. In addition, as the GP ratio increases, the time for LVT computation, event insertion, etc. may increase.

As the gate-to-processor ratio increases, each processor contains more gates. That is, memory space allocated to a gate decreases. We cannot implement simulation techniques on current massively parallel SIMD machines to get accurate experimental results for large gate-to-processor ratio since the machines have memory size problems. When a massively parallel SIMD machine with enough local memory becomes available, actual execution times on the machine can be measured.

Parallel VHDL Simulator

In [17], we have studied a parallel VHDL simulator using parallel logic simulation. The VHDL description for a circuit is translated into an intermediate form for simulation. The intermediate form is simulated with discrete event simulation techniques, such as Time Warp or the Chandy-Misra algorithm, on the Connection Machine. Signal assignment statements in the VHDL description are transformed into gate processors, and component instantiations are handled by their expansion into a combination of basic elements. As future study, we consider more extensive and practical low level interfaces for a parallel VHDL simulator.

We plan to include behavioral descriptions and to extend the set of available data types. We also plan to evaluate the use of other paradigms available for parallel VHDL simulation, such as process-oriented Time Warp schemes and the Chandy-Misra algorithm. Note that our simulation scheme, which decomposes the circuits into basic gate elements and runs the simulation specification on a parallel machine, can also be applied to other types of parallel machines such as the BBN Butterfly, a shared memory MIMD machine. Hence performance on other machine architectures will also be explored.

Bibliography

Bibliography

[1] *VHDL Language Reference Manual*, 1987.

[2] M. Abramovici, Y. H. Levendel, and P. R. Menon. A logic simulation machine. In *Proceedings of the 19th Design Automation Conference*, pages 65–73. ACM/IEEE, 1982.

[3] V. D. Agrawal and S. T. Chakradha. Performance estimation in a massively parallel system. In *Proceedings of the Supercomputing'90*, pages 306–313. ACM/IEEE, November 1990.

[4] J. R. Agre. Simulations of time warp distributed simulations. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 85–90, March 1989.

[5] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing*, pages 301–351. Addison-Wesley, 1989.

[6] R. Ayani. A parallel simulation scheme based on the distance between objects. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 113–118, March 1989.

[7] R. Baldwin, M. J. Chung, and Y. Chung. Overlapping window algorithm for computing GVT in Time Warp. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 534–541. IEEE, May 1991.

[8] D. Ball and S. Hoyt. The adaptive Time Warp concurrency control algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 174–177, January 1990.

[9] BBN Advanced Computers Inc. *BBN GP1000 Switch Tutorial*, March 1989.

[10] O. Berry and D. R. Jefferson. Critical path analysis of distributed simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 57–60, January 1985.

[11] T. Blank. The MasPar MP-1 architecture. In *Proceedings of the 35th IEEE COMPCOM Spring 1990*, pages 20–24, February 1990.

[12] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proceedings of International Symposium on Circuits and Systems*, pages 1929–1934. IEEE, May 1989.

[13] F. Brglez, P. Pownall, and R. Hum. Accelerated ATPG and fault grading via testability analysis. In *Proceedings of International Symposium on Circuits and Systems*, pages 695–698. IEEE, June 1985.

[14] J. Briner. *Parallel Mixed-Level Simulation of Digital Circuits Using Virtual Time.* PhD thesis, Duke University, 1990.

[15] R. E. Bryant. Data parallel switch-level simulation. In *Proceedings of the 1988 International Conference on Computer Aided Design*, pages 354–357, 1988.

[16] R. E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler. COSMOS: A compiled simulator for MOS circuits. In *Proceedings of the 24th Design Automation Conference*, pages 9–16. ACM/IEEE, 1987.

[17] A. D. Cabrera, M. J. Chung, and Y. Chung. A parallel VHDL simulator on the Connection Machine. Technical report, Department of Computer Science, Michigan State University, October 1990.

[18] A. Chandak and J. C. Browne. Vectorization of discrete-event simulation. In *Proceedings of the 1983 International Conference on Parallel Processing*, pages 359–361, August 1983.

[19] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, 5(5):440–452, September 1979.

[20] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–206, April 1981.

[21] K. M. Chandy and J. Misra. *Parallel Program Design, A Foundation.* Adison-Wesley, 1988.

[22] K. M. Chandy and R. Sherman. The conditional event approach to distributed simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 93–99, March 1989.

[23] E. M. Choi, M. J. Chung, and Y. Chung. Comparisons and analysis of massively parallel SIMD architectures for parallel logic simulation. In *Proceedings of the Sixth International Parallel Processing Symposium*, March 1992.

[24] M. J. Chung and Y. Chung. Data parallel simulation using Time Warp on the Connection Machine. In *Proceedings of the 26th Design Automation Conference*, pages 98–103. ACM/IEEE, June 1989.

[25] M. J. Chung and Y. Chung. Efficient parallel logic simulation techniques for the Connection Machine. In *Proceedings of the Supercomputing'90*, pages 606–614. ACM/IEEE, November 1990.

[26] M. J. Chung and Y. Chung. A distributed token-driven technique for parallel zero-delay logic simulation on massively parallel machines. In *Proceedings of 1991 International Conference on Parallel Processing*, pages 391–394, August 1991.

[27] M. J. Chung and Y. Chung. An experimental analysis of simulation clock advancement in parallel logic simulation on an SIMD machine. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 125–132, January 1991.

[28] M. J. Chung and Y. Chung. Parallel logic simulation on a massively parallel SIMD machine. Technical report, Department of Computer Science, Michigan State University, October 1991.

[29] Y. Chung and M. J. Chung. Time Warp for efficient parallel logic simulation on a massively parallel SIMD machine. In *Proceedings of the Tenth Annual International Phoenix Conference on Computers and Communications*, pages 183–189. IEEE, March 1991.

[30] R. C. De Vries. Reducing null messages in Misra's distributed discrete event simulation model. *IEEE Transactions on Software Engineering*, 16(1):82–91, January 1990.

[31] J. R. Engelsma, M. J. Chung, and Y. Chung. Distributed token-driven logic simulation on a shared-memory multiprocessor. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, pages 197–198. ACM/IEEE/SCS, January 1992.

[32] R. M. Fujimoto. Lookahead in parallel discrete event simulation. In *Proceedings of 1988 International Conference on Parallel Processing*, volume 3, pages 34–41, 1988.

[33] R. M. Fujimoto. Performance measurements of distributed simulation strategies. *Transactions of the Society for Computer Simulation*, 6.2:89–132, April 1989.

[34] R. M. Fujimoto. Time Warp on a shared memory multiprocessor. In *Proceedings of 1989 International Conference on Parallel Processing*, volume 3, pages 242–249, 1989.

[35] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[36] R. L. Geiger, P. E. Allen, and N. R. Strader. *VLSI Design Techniques for Analog and Digital Circuits*, pages 909–917. McGraw-Hill, 1990.

[37] J. B. Gilmer. An assessment of Time Warp parallel discrete event simulation algorithm performance. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 45–49, July 1988.

[38] J. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31:523–533, May 1988.

[39] H. Hansen. Hardware logic simulation by compilation. In *Proceedings of the 25th Design Automation Conference*, pages 712–715. ACM/IEEE, 1988.

[40] J. P. Hayes. *Computer Architecture and Organization*, pages 187–190. McGraw-Hill, 1978.

[41] D. R. Jefferson. Fast concurrent simulation using the Time Warp mechanism. In *Proceedings of the 1985 Multiconference on Distributed Simulation*, pages 63–69. SCS, January 1985.

[42] D. R. Jefferson. Virtual time. *ACM Trans. Programming Languages and Systems*, 7(3):404–425, July 1985.

[43] D. R. Jefferson et al. Implementation of Time Warp on the Caltech Hypercube. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 70–75, January 1985.

[44] Y. Lan. *Interprocessor Communication in Distributed Memory Multiprocessors.* PhD thesis, Michigan State University, 1988.

[45] Y. H. Levendel, P. R. Menon, and S. H. Patel. Special-purpose computer for logic simulation using distributed processing. *The Bell System Technical Journal*, 61(10):2873–2909, December 1982.

[46] Y. Lin and E. D. Lazowaska. Determining global virtual time in a distributed simulation. Technical report, 90-01-02, Department of Computer Science, University of Washington, December 1989.

[47] Y. Lin and E. D. Lazowaska. Reducing the state saving overhead for Time Warp parallel simulation. Technical report, 90-01-03, Department of Computer Science, University of Washington, February 1990.

[48] Y. Lin, E. D. Lazowaska, and M. L. Bailey. Comparing synchronization protocols for parallel logic-level simulation. In *Proceedings of the 1990 International Conference on Parallel Processing*, volume 3, pages 223–227, August 1990.

[49] B. D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. *Communications of the ACM*, 32(1):111–123, January 1989.

[50] V. Madisetti, J. Walrand, and D. Messerschmitt. WOLF: A rollback algorithm for optimistic distributed simulation systems. In *Proceedings of the 1988 Winter Simulation Conference*, December 1988.

[51] MasPar Computer Corporation. *MasPar MP-1*, July 1990.

[52] P. M. Maurer and Z. Wang. Techniques for unit-delay compiled simulation. In *Proceedings of the 27th Design Automation Conference*, pages 480–484. ACM/IEEE, 1990.

[53] W. Myers, Editor. Massively parallel systems break through at supercomputing '90. *IEEE Computer*, 24(1):121–126, January 1991.

[54] J. K. Peacock, J. W. Wong, and E. G. Manning. Distributed simulation using a network of processors. *Comput. Networks*, 3(1):44–56, February 1979.

[55] D. L. Pitts and D. L. Smith. Central concurrency control for distributed simulations. In *Proceedings of 1988 Summer Computer Simulation Conference*, 1988.

[56] B. R. Preiss. The Yaddes distributed discrete event simulation specification language and execution environment. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 139–144, January 1989.

[57] B. R. Preiss. Performance of discrete event simulation on a multiprocessor using optimistic and conservative synchronization. In *Proceedings of the 1990 International Conference on Parallel Processing*, pages 218–222, August 1990.

[58] D. A. Reed, A. D. Malony, and B. D. McCredie. Parallel discrete event simulation using shared memory. *IEEE Transaction on Software Engineering*, 14(4):541–553, April 1988.

[59] A. E. Ruehli and G. S. Ditlow. Circuit analysis, logic simulation, and design verification for VLSI. *Proceedings of the IEEE*, 71(1):34–48, January 1983.

[60] S. P. Smith, M. R. Mercer, and B. Brock. Demand driven simulation:BACKSIM. In *Proceedings of the 24th Design Automation Conference*, pages 181–187. ACM/IEEE, 1987.

[61] L. M. Sokol, B. K. Stucky, and V. S. Hwang. MTW: A control mechanism for parallel discrete simulation. In *Proceedings of the 1989 International Conference on Parallel Processing*, volume 3, pages 250–254, August 1989.

[62] L. Soule and T. Blank. Parallel logic simulation on general purpose machines. In *Proceedings of the 25th Design Automation Conference*, pages 166–171. ACM/IEEE, June 1988.

[63] L. Soule and A. Gupta. Characterization of parallelism and deadlocks in distributed digital logic simulation. In *Proceedings of the 26th Design Automation Conference*, pages 81–86. ACM/IEEE, June 1989.

[64] J. S. Steinman. SPEEDES:synchronous parallel environment for emulation and discrete event simulation. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 95–103, January 1991.

[65] R. E. Tarjan and D. D. Sleator. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.

[66] Thinking Machines Corporation. *The Connection Machine System*, May 1988.

[67] G. Varghese and T. Lauck. Hashed and hierarchical timing wheels: Data structures for the efficient implementation of a timer facility. In *Proceedings of the Eleventh Symposium on Operating System Principles*, pages 25–33. ACM, November 1987.

[68] A. H. Veen. Dataflow machine architecture. *ACM Computing Surveys*, 18(4):365–396, December 1986.

[69] L. Wang et al. SSIM: A software levelized compiled-code simulator. In *Proceedings of the 24th Design Automation Conference*, pages 2–8. ACM/IEEE, 1987.

[70] Z. Wang and P. M. Maurer. LECSIM: A levelized event driven compiled logic simulator. In *Proceedings of the 27th Design Automation Conference*, pages 491–496. ACM/IEEE, 1990.

[71] D. M. Webber and A. Sanggiovanni-Vincentelli. Circuit simulation on the Connection Machine. In *Proceedings of the 24th Design Design Automation Conference*, pages 108–113. ACM/IEEE, June 1987.

[72] N. Weste and K. Eshraghian. *Principles of CMOS VLSI design*, pages 255–256. Adison-Wesley, 1985.

[73] H. Xu, P. K. Mckinley, and L. M. Ni. Efficient implementation of barrier synchronization in wormhole routed hypercube multicomputers. Technical report, MSU-CPS-ACS-47, Department of Computer Science, Michigan State University, October 1991.

[74] M. Yu, S. Ghosh, and E. DeBenedictis. A non-deadlocking conservative asynchronous distributed discrete event simulation algorithm. In *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 39–43. ACM/IEEE/SCS, January 1991.