1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**Time Warping - Waveform Relaxation (TW - WR)
in a Distributed Simulation Environment**

by

Carlos Pon, M.Eng.

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree of**

Doctor of Philosophy

**Department of Electronics
Faculty of Engineering
Carleton University
February 1995.**

© copyright

1995, Carlos Pon

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-612-02960-3

Canada

Name

Charles Pon

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

0544

U·M·I

SUBJECT TERM

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0273
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern 0291
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European 0370

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology 0323

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0578
Geography 0366
Gerontology 0351
History 0578
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography 0628
Ethnic and Racial Studies 0629
Individual and Family Studies 0630
Industrial and Labor Relations 0700
Public and Social Welfare 0709
Social Structure and Development 0999
Theory and Methods 0453
Transportation 0453
Urban and Regional Planning 0453
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and Nutrition 0476
Animal Pathology 0359
Food Science and Technology 0478
Forestry and Wildlife 0479
Plant Culture 0480
Plant Pathology 0817
Plant Physiology 0777
Range Management 0746
Wood Technology 0306
Biology 0287
General 0308
Anatomy 0309
Biostatistics 0379
Botany 0329
Cell 0353
Ecology 0369
Entomology 0793
Genetics 0410
Limnology 0307
Microbiology 0317
Molecular 0416
Neuroscience 0433
Oceanography 0821
Physiology 0778
Radiation 0472
Veterinary Science 0786
Zoology 0760
Biophysics 0786
General 0760
Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology 0354
Occupational Health and Therapy 0381
Ophthalmology 0571
Pathology 0419
Pharmacology 0572
Pharmacy 0382
Physical Therapy 0573
Public Health 0574
Radiology 0575
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics 0605
Physics 0986
General 0606
Acoustics 0608
Astronomy and Astrophysics 0748
Atmospheric Science 0607
Atomic 0798
Electronics and Electricity 0759
Elementary Particles and High Energy 0609
Fluid and Plasma 0610
Molecular 0752
Nuclear 0756
Optics 0611
Radiation 0463
Solid State 0346
Statistics 0984
Applied Sciences 0346
Applied Mechanics 0984
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 351
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology 0796
Operations Research 0795
Plastics Technology 0994
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned recommend to the Faculty of Graduate
Studies and Research acceptance of the thesis

**TIME WARPING - WAVEFORM RELAXATION (TW - WR) IN A DISTRIBUTED
SIMULATION ENVIRONMENT**

submitted by Carlos Pon
in partial fulfilment of the requirements for
the degree of Doctor of Philosophy

Chair, Department of Electronics

Thesis Supervisor

External Examiner

Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Faculty of Engineering
Carleton University
April 1995

Abstract

Circuit simulation at the electrical level continues to be the bottleneck in the VLSI design and verification process. Parallel and distributed processing is considered as a viable approach to improve the performance of circuit simulators. In this thesis, two methods of waveform relaxation (WR) distributed circuit simulation are considered.

In the first method, each processor can simulate along a selected time window, and the processors never get ahead unless all other subcircuits allocated in other processors have converged in the current time window (Conservative WR). In the other method, processors are not concerned about future arriving events. They simulate independently and roll back only if an event arrives that belong to the past. This new, proposed method referred as the Time Warping-Waveform Relaxation (TW-WR) shows 1.6 to 3.2 improvement speedups over the conservative method when the simulator is ported on a network of eight Sun Sparc-2 workstations.

Acknowledgments

The author wishes to thank with deep appreciation his thesis supervisor, professor Tad Kwasniewski for his professional guidance and moral support through the extent of this thesis.

The author is also indebted to professor Miles Copeland for the financial support of this project through TRIO (Telecommunication Research Institute of Ontario).

Also, I would like to thank Dr. Resve Saleh from the University of Illinois, Urbana-Champaign, for giving me the opportunity to get my hands in his parallel environment and for providing the necessary tools to carry this thesis to completion.

Dan Withey, also deserves special acknowledgment for his collaboration in the implementation of the interprocess communication routines for the slotted ring architecture.

Last but not least, I would like to thank my wife Sonia and my children Chantal, Anthony and Charlotte for their strong support during this study.

A mis padres, Guillermo y Elena

TABLE OF CONTENTS

List of Figures	x
List of Tables	xiii
Abbreviations	xiv
I Introduction	1
1.1 Why Distributed Circuit Simulation?	4
1.2 Scope of the Thesis	5
1.3 Main Contributions	7
II Standard Circuit Simulation and Decomposition Methods	8
2.1 Advantages and Disadvantages of Standard Circuit Simulation	8
2.2 Tearing Methods	12
2.3 Relaxation Based Methods	14
2.3.1 Linear Relaxation	15
2.3.2 Non-Linear Relaxation Methods	17
2.3.3 Waveform Relaxation Method	19
2.3.4 ITA and WR Algorithm Speedups	24
2.4 Conclusions	28
III Parallel Architectures and Algorithms in the Acceleration of Circuit Simulation	30
3.1 Parallel Processing System Classification	31
3.1.1 Shared-Memory Systems	33
3.1.2 Distributed Memory Systems	38
3.2 A Multiprocessor System Interconnected in a Ring Topology	40
3.2.1 Iterative Solution of Dense Linear Equations	41
3.2.2 Performance Study in a Slotted Ring Topology	43
3.3 Parallel Processing Circuit Simulation using Direct Methods	47
3.3.1 Parallel Model Evaluation	50

3.3.2	Standard Parallel Sparse Matrix Solution	51
3.3.3	Parallel Circuit Simulation using Tearing Methods	52
3.3.4	Parallelism issues for Direct Methods	55
3.4	Parallel Processing in Relaxation Methods	55
3.4.1	Parallelism in Iterative Timing Analysis	58
3.4.2	Parallelism in Waveform Relaxation	64
3.5	Waveform Relaxation in a Distributed Simulation Environment	70
3.6	Conclusions	72

IV Spatial and Temporal Partitioning in a WR Distributed Simulation Environment 74

4.1	Spatial and Temporal Partitioning in the Waveform Relaxation	74
4.1.1	Spatial Domain Decomposition	75
4.1.1.1	DC Unilateral Subcircuit Partitioning	76
4.1.1.2	Norton Admittance Partitioning	77
4.1.1.3	Dynamic Partitioning	80
4.1.1.4	Hierarchical Partitioning	80
4.1.1.5	Load Balance - Convergence Trade-off Partitioning	82
4.1.2	Temporal Domain Decomposition	85
4.1.2.1	Fixed Time Window Decomposition	85
4.1.2.2	Dynamic Window Decomposition	91
4.2	Task Granularity in a WR Distributed Environment	91
4.3	A Portable Parallel Simulator	94
4.4	Summary	96

V WR Parallelism in the Spatial and Temporal Domain, Scheduling and Task Allocation 98

5.1	Subcircuit Ordering	98
5.2	Task Parallelism and Scheduling in the Spatial Domain	100
5.2.1	Data-Flow Based Algorithm	101
5.3	Task Parallelism and Scheduling in the Temporal Domain.	105
5.4	Time-Segment Pipelining	108

5.5	Task Allocation in a Ring Architecture	112
5.6	Conclusions	115
VI	Conservative Synchronization in a WR Distributed Simulation Environment	117
6.1	A Conservative WR in a Distributed Simulation Environment	117
6.2	Critical Path Analysis in the Conservative WR Distributed Simulation ..	122
6.3	Conservative WR Simulator Performance on Parallel Architectures	125
6.4	Distributed Circuit Simulation WR Performance in the Sun Network	133
6.5	TSPL Analysis in a Distributed Circuit WR Simulation	136
6.6	Summary	140
VII	Time Warping in the Waveform Relaxation Distributed Circuit Simulation Environment	141
7.1	Main Limitations for Conservative Distributed WR Methods	141
7.2	The Time Warping Mechanism applied to Waveform Relaxation	144
7.2.1	Antimessages	145
7.2.2	Aggressive Cancellation	146
7.2.3	Lazy Cancellation	146
7.2.4	Data Structure of a Subcircuit in TW-WR	146
7.2.5	Global Virtual Time	148
7.3	Graphical Example of Two Inverters Simulation using TW-WR	148
7.4	Critical Path Analysis in the TW-WR Distributed Circuit Simulation	150
7.5	TW-WR Implementation	156
7.6	Experimental Results	160
7.7	Conclusions	167
VIII	TW-WR Simulator Performance	169
8.1	TW-WR Simulator Performance on a Chain of 8 Inverters	169
8.2	TW-WR Performance for Commercial Benchmark Circuits	173
8.3	Potential Concurrency for Circuit Examples using TW-WR	182
8.4	Conclusions	183

IX	Conclusions	184
9.1	Future Directions	186
	References	187
	Appendix A:Solution Steps of Circuit Simulation using DM	197
	Appendix B:Branch and Node Tearing in Circuit Simulation Decomposition	203
	Appendix C:A Slotted-Ring Bus Architecture Description and its Performance	207

List of Figures

FIGURE 1	Solution of equations by a conventional circuit simulator	10
FIGURE 2	SPICE execution profile of a VLSI memory circuit with 3700 equations ..	11
FIGURE 3	BBD and BBT matrix structures	13
FIGURE 4	Solution of equations with use of relaxation methods	14
FIGURE 5	(a) Logic representation of a dynamic shift register	22
FIGURE 6	Circuit decomposition from figure 5 using waveform relaxation	23
FIGURE 7	Graphical comparison in the Gauss-Seidel iteration	24
FIGURE 8	Partial waveform convergence with two iterations	27
FIGURE 9	SIMD execution	32
FIGURE 10	Bit-plane array processing	33
FIGURE 11	Shared memory configuration	34
FIGURE 12	MIMD shared-memory interconnection schemes:	36
FIGURE 13	Distributed memory configuration	38
FIGURE 14	MIMD interconnection network topologies:	39
FIGURE 15	The hypercube topology	40
FIGURE 16	Iteration times for the theoretical matrix solver (solid-line)	45
FIGURE 17	Speedup time ratios for a single iteration in the matrix solver	46
FIGURE 18	Communication overhead for the iteration matrix solver using 4	46
FIGURE 19	Amdahl's law graphical representation	48
FIGURE 20	(a) BBD structure (b) nested RBBD structure	53
FIGURE 21	Task graph representation of the RBBD structure of figure 20 .	54
FIGURE 22	Examples of Gauss-Seidel and Gauss-Jacobi partial ordering	57
FIGURE 23	Dynamic decoupling using two phase clock cycles in a chain of inverters .	65
FIGURE 24	The three level scheduler	68
FIGURE 25	Illustration of spatial partitioning	75
FIGURE 26	Generic coupling between two nodes in a circuit	77
FIGURE 27	Linear circuit considered for partitioning	78

FIGURE 28	Algorithm for conductance partitioning	79
FIGURE 29	Hierarchical spatial partitioning with 2 layers	81
FIGURE 30	WR program structure	86
FIGURE 31	Run times for a chain of N inverters using WR, iSPICE3	87
FIGURE 32	Run times for a ring oscillator with a variable number of	89
FIGURE 33	Run times for a ring oscillator with a variable number of	90
FIGURE 34	Speedup ratios between HSPICE/WR (solid lines) and	90
FIGURE 35	Algorithm framework on the portable parallel simulator	95
FIGURE 36	Examples of Gauss-Seidel and Gauss-Jacobi partial ordering	99
FIGURE 37	Task dependency graph for the overlapped WR iteration algorithm	103
FIGURE 38	(a) Basic structure of a four pipeline processor	106
FIGURE 39	A sequential linear circuit with three subcircuits	108
FIGURE 40	Task dependency graph for the TSPL algorithm	110
FIGURE 41	Task-time diagram showing the available parallelism in TSPL	111
FIGURE 42	Task allocation graph in a distributed ring architecture	113
FIGURE 43	Parallel WR pseudo-code	119
FIGURE 44	(a) Unordered group of processors in a network	120
FIGURE 45	Pseudo-code for global synchronization in a distributed	121
FIGURE 46	Chain of four inverters and partial ordering graph with their	122
FIGURE 47	Sequence of events generated for a chain of four inverters	124
FIGURE 48	Sequence of events generated for a chain of four inverters	126
FIGURE 49	Speedup curves versus number of PEs for the benchmark circuits	128
FIGURE 50	Speedup curves versus number of PEs for the benchmark circuits	130
FIGURE 51	Runtime contributions for different synthetic benchmark circuits	132
FIGURE 52	Speedup curves versus number of PEs for commercial circuits	134
FIGURE 53	Runtime contributions for commercial benchmark circuits	136
FIGURE 54	Sequence of events generated by a time segment pipelining in WR	137
FIGURE 55	Three dimensional domain parallelism in WR: spatial, temporal	139
FIGURE 56	Two inverters subcircuits simulation using the time warping	149

FIGURE 57	Message-events generated in the input and output queues	151
FIGURE 58	Sequence of events generated for a two inverters simulation using	152
FIGURE 59	Sequence of events generated for a chain of four inverters in	154
FIGURE 60	Relative speedup performance of TW-WR with respect to the	155
FIGURE 61	CPU runtime for the conservative and TW-WR versus the number	156
FIGURE 62	Graphical data structures for the subcircuit's waveform buffers	158
FIGURE 63	Pseudo-code for the GVT computation	159
FIGURE 64	GVT calculation before and after synchronization	159
FIGURE 65	Speedup curves versus number of PEs for the benchmark circuits	162
FIGURE 66	Bar graph display of all the contributions for the synthetic	163
FIGURE 67	Speedup curves versus the number of PEs for the commercial	165
FIGURE 68	Runtime contributions for the commercial circuit benchmarks	166
FIGURE 69	Runtime accumulation versus window instance for the inv8	170
FIGURE 70	CPU runtime plus communication overhead versus the number	171
FIGURE 71	Total runtime versus number of PEs for the circuit of 8	172
FIGURE 72	Total CPU runtime versus number of PEs, using different	173
FIGURE 73	CPU runtime versus number of PEs, considering CPU simulation	175
FIGURE 74	CPU runtime versus number of PEs, considering CPU simulation	176
FIGURE 75	Total CPU runtime versus number of PEs using the TW-WR	177
FIGURE 76	Number of rollbacks versus the number of PEs, using the	178
FIGURE 77	Performance improvement of the TW-WR over the	179
FIGURE 78	Improved speedup curves of the TW-WR algorithm over the	179
FIGURE 79	CPU runtimes versus the number of PEs for the commercial	180
FIGURE 80	Amount of memory used by the TW-WR simulator (solid lines) and	181
FIGURE 81	Upper bound speedups for the commercial circuits assuming	182
FIGURE 82	Partitioned network and its torn representation	204
FIGURE 83	Network partitioning using node tearing	206
FIGURE 84	Unidirectional slotted ring bus	207
FIGURE 85	Input/output processor description block	208

List of Tables

TABLE 1	Speedup obtained in the slotted-ring architecture with 4 PEs for	44
TABLE 2	Scdac runtimes and circuit partitioning using different threshold	82
TABLE 3	Iterative circuit partitioning using and	84
TABLE 4	Average time point simulation for different benchmark circuits	93
TABLE 5	Lower bound computation/communication metric ratio	94
TABLE 6	Circuit Load distribution for a system with 4 processors	114
TABLE 7	WR runtimes and speedups in the slotted-ring architecture	127
TABLE 8	WR runtimes and speedups in Sun Sparc-2 workstations	129
TABLE 9	WR simulation, communication and synchronization overhead in	131
TABLE 10	Experimental runtimes and speedup for commercial circuits using a	133
TABLE 11	WR simulation, communication and synchronization overhead in	135
TABLE 12	Simulation runtimes and speedup with respect to the conservative	155
TABLE 13	TW-WR runtimes and Speedups for the synthetic benchmark	161
TABLE 14	WR simulation, communication and synchronization overhead in	163
TABLE 15	Experimental runtimes and speedups for commercial circuits using	164
TABLE 16	WR simulation, communication and synchronization overhead in	167
TABLE C.1.	Latency for Furthest Intercommunication Path in a N PE System	211

Abbreviations

ACK	Acknowledge
BBD	Bordered Block Diagonal
BBT	Bordered Block Traingular
BDF	Backward Differentiation Formula
CAD	Computer Aided Design
CP	Cross-point Switches
CPU	Central Processing Unit
DC	Direct Current
DF	Data Flow
DM	Direct Methods
DPRAM	Dual Port RAM
DSP	Digital Signal Processing
ED	Event Driven
FET	Field Effect Transistor
FIFO	First In First Out
G-J	Gauss-Jacobi
G-S	Gauss-Seidel
GVT	Global Virtual Time
GW	Ground Wave
IC	Integrated Circuit
I/O	Input/Output
IOP	Input-Output Processor
IN	Interconnection Network
ITA	Iterative Time Analysis
LDU	Local Datum Nodes
LTE	Local Truncation Error
LVT	Local Virtual Time

LU	Lower and Upper
MIMD	Multiple Instruction Multiple Data
MIN	Multistage Interconnection Network
MISD	Multiple Instruction Single Data
MNA	Modified Nodal Analysis
MOS	Metal Oxide Semiconductor
NMOS	N type MOS
ODE	Ordinary Differential Equations
OWRI	Overlap Waveform Relaxation Iteration
PE	Processor Element
PLA	Programmable Logic Array
PME	Parallel Model Evaluation
PWC	Partial Waveform Convergence
RAM	Random Access Memory
RBBD	Recursive BBD
RHS	Right Hand Side
SCC	Strongly Connected Components
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
TCP/IP	Transport Communication Protocol/ Internet Protocol
TPP	Time Point Pipelining
TSPL	Time Segment Pipelining
TW	Time Warping
VLSI	Very Large Scale Integration
WR	Waveform Relaxation

I Introduction

The simulation of integrated circuits at the detailed electrical level continues to be an indispensable element in the overall IC design and verification process. However, the dramatic increase in the complexity of IC's over the last years has strained the capabilities of traditional circuit simulators such as SPICE [38] and ASTAP [69].

A Sun Sparc-2 workstation can take several hours or days to simulate circuits that contain a few thousand transistors at the electrical level, and this simulation problem has become more serious with today's chip densities. With today's technology it is possible to pack a few million transistors in a chip, and device densities continue to grow at a faster rate than the performance of the computers that can simulate them; designers agree this is a bottleneck limiting their productivity. This situation has resulted in the creation of different approaches to obtain acceleration in the simulation process for VLSI circuits, involving non-standard hardware resources and new algorithmic methods. Many simulators have evolved, trading accuracy for speed, which has lead to a number of higher simulation tools to cope with the complexity of the problem. These tools include behavioral simulators, register-transfer-level (RTL) simulators, gate level logic simulators, switch level simulators and timing simulators. Although these tools are no doubt useful, none of them is considered as reliable, accurate and technology independent as the electrical level circuit simulator.

Modern simulators incorporate more than one hierarchy level so that models at different levels can be cross-checked and each simulation level can provide successively more detailed information, at the expense of increased computational cost. These mixed-mode simulators are especially useful in the design of a mixed analog and digital circuit, and for top-down or bottom-up design environments. When accuracy is not a critical issue, higher levels of simulation can be used, but when accurate circuit performance is required, lower levels of simulation are required. Nevertheless, even if a small percentage of the circuit (10 to 20%), i.e. the critical path of the circuit, is being used at the electrical level in a mixed-mode simulator, this bottom level of simulation still represents a major bottleneck in the simulation process, because the ratio of complexity of different levels is still of several orders of magnitude

The standard approach to the numerical solution for the circuit simulation problem, called Direct Methods (DM) has proven to be reliable and accurate, but its sparse-matrix solution time grows super-linearly with the size of the circuit, resulting in unrealistically long simulation times for complex VLSI circuits. The other drawback of the DM is inherent to the direct application of the integration method, forcing every differential equation in the system to be discretized with identical time steps. Such a discretization must be fine enough so that the fastest-changing circuit variable in the system can be accurately represented.

The new generation of programs labelled “Relaxation Methods”, has been proposed to alleviate these problems. The Iterated Timing Analysis (ITA) algorithm [53] uses a relaxation scheme at the non-linear algebraic equation level. This technique, in conjunction with an event-driven selective trace algorithm, exploits the latency or relative inactivity of the nodes of the circuit.

In the Waveform Relaxation (WR) method [35], the relaxation is applied at the

differential equation level. This method exploits multirate properties, since circuit decomposition allows the differential equations to be solved with different time steps.

Both methods have guaranteed convergence when the circuits meet certain restrictive conditions. Although these techniques claim a simulation speedup of up to two orders of magnitude, this still is not adequate to efficiently simulate the circuits of increasing size of VLSI designs. The magnitude of the task has lead to the development of new techniques that rely on usage of hardware resources. Today, the advances of parallelism in computer technology are such that multiprocessors provide enormous computational power at a reasonable cost. Numerous applications have been implemented on both special purpose accelerators and general purpose machines.

Therefore, it is considered that a viable approach to improve the performance of VLSI circuit simulation is the use of parallel and distributed processing. Much attention has been focused in the last ten years on the mapping of circuit simulation algorithms to parallel architectures which use the shared-memory paradigm to facilitate parallel computation [10,77,21,23,27,30,45,51,56]. With the advent of high performance workstations and high speed interconnection networks, distributed processing offers a cost effective and attractive solution to the circuit simulation problem. Workstations available at universities and research centers can be effectively used for distributed simulation, resulting in shorter simulation times with no additional investment. As well, workstations are more general purpose computers and are much easier to use and hence can be used for a wider variety of applications as compared to parallel dedicated machines. The major drawback of using such a network of workstations in circuit simulation is the high overhead involved in the interprocessor communication, and therefore, a careful analysis is needed. The main subject explored in this dissertation is the use of distributed algorithms for the acceleration of circuit simulation. Therefore, the distributed algorithms

considered must guarantee efficient use of the hardware resources. This can be achieved by partitioning the problem and allocating them through the system, and minimizing their interactions.

1.1 Why Distributed Circuit Simulation?

Parallel simulation has often been suggested as a means of speeding up circuit simulation. In the near future, is expected that a thousand processors elements (PE) will be connected through a very high speed interconnection network, in which each PE can communicate directly with a few other PEs. A simulation model in such a distributed environment is then an entire circuit partitioned into loosely coupled subcircuits where each one can be simulated concurrently. Distributed simulation seems to be an attractive solution, because subcircuits often interact weakly and can be simulated in parallel, this being particularly true for very large circuits. Distributed circuit simulation is also very attractive, since time and memory constraints can be reduced by distributing the load among all the processors. As well, distributed algorithms are scalable to the problems, meaning they can run on a number of processors determined by problem complexity.

Therefore, the distributed techniques used for the circuit simulation environment require a set of processors that can communicate by sending messages along the links of a communication network. The processors each simulate a subcircuit or a collection of them and interact when necessary.

Two methods of waveform relaxation (WR) distributed circuit simulation are considered in this thesis. Both methods are fully distributed in the sense that they require no central control. In the first method considered, each processor can simulate independently along a selected time window, and the processors never get ahead (lead in time) unless all other subcircuits allocated in other processors have converged in the current time window.

This method is referred in this dissertation as Conservative WR. In the second method, processors are not concerned about future arriving events. They simulate independently and roll back only if an event arrives that belongs to the past. This new, proposed method is referred to here as the Time Warping Waveform Relaxation technique (TW-WR).

1.2 Scope of the Thesis

Relaxation decomposition methods are viable solutions for the acceleration of circuit simulation in a distributed environment. Iterative Timing Analysis in conjunction with an event-driven selective trace algorithm, has been mapped successfully in different types of architectures such as hypercubes, transputers, shared memory, and hybrid. This ITA technique benefits from the latency of subcircuits and is known as a medium size grain computation (as compared to the large grain computation waveform relaxation technique). The task size in waveform relaxation technique is much larger, since it is based on a collection of data points of ITA type. WR technique has been adopted in this dissertation since it can exploit more efficiently the multirate properties existing in a circuit and also since the task grain size is sufficiently large to be ported to a network of workstations. Both relaxation algorithms are described in Chapter II, where they are compared to the standard circuit simulation methods.

Chapter III begins with a presentation of commercial dedicated hardware that has been used over the past few years to accelerate circuit simulation. A dedicated slotted-ring architecture is also presented in this chapter, which has been selected as dedicated hardware to test the circuit simulation distributed algorithms. Parallel circuit simulation algorithms are presented next. The chapter concludes with the framework of distributed circuit waveform relaxation environment.

The grain size determination in a distributed architecture depends primarily on the partitioning method used. Chapter IV focuses on this issue by decomposing the simulation model of waveform relaxation in the spatial and temporal domain. Finally task granularity is determined for the two selected architectures used as the testbed of our circuit simulation algorithms.

Chapter V discusses possible techniques to exploit the task parallelism in the waveform relaxation algorithm. Data-flow and event-driven techniques are presented as asynchronous methods which exploit the functional parallelism in the spatial and iteration domains. Time segment pipelining is also presented as a method to explore parallelism in the temporal domain. Finally, an heuristic task allocation algorithm for the slotted ring architecture is included to exploit parallelism in all domains. This algorithm is based on the critical path of the dependency task graph.

Chapters VI and VII concentrate on the two methods for distributed circuit simulation. Both methods use discrete event circuit simulation using waveform relaxation techniques. In Chapter VI, the circuit simulator is based on a conservative synchronization method. As mentioned earlier in this method each processor can simulate independently as long as it is certain that no past events will arrive. Experimental results are presented for the two selected architectures. In Chapter VII, a synchronization method and experimental results are presented which allow processors to simulate 'ahead of time'. If any past event arrives, the processors adjust their operations by rolling the simulation time back to ensure a consistent state. Experimental results are presented in Chapters VI and VII that use the same set of benchmark circuits. One consist of synthetic, benchmark circuits and the other consists of industrial circuits. A critical path analysis is used in both chapters to analyze the potential concurrency. A chain of inverters is used as a test circuit.

Chapter VIII presents the overall TW-WR simulator performance over the Conservative WR using the network of Sun Sparc-2 workstations on a Ethernet bus. Finally the potential speedup is presented for networks with higher communication bandwidths.

1.3 Main Contributions

The contributions of this dissertation include:

1. Identification of bottlenecks of the conservative WR distributed circuit simulation algorithm in a network of workstations.
2. A heuristic, static subcircuit allocation algorithm for a dedicated slotted ring architecture implementation based on the conservative WR technique. Speedup measurements were obtained for four processors.
3. A Time Warping mechanism is introduced in the WR distributed circuit simulation environment to alleviate the bottlenecks of the conservative WR method. The Time Warping technique is a modified one, with no cancellation.
4. Evaluation results supporting a claim that TW-WR can exploit parallelism in the temporal domain as well as the spatial domain, and give higher speedups than the lower boundary found in the conservative method.
5. The main contribution of this thesis is the development and verification of a new distributed simulation algorithm (TW-WR) characterized by a relative insensitivity to the interprocessor communication overhead. It is demonstrated that the speedups obtained using a distributed simulation algorithm in a network of Sun workstations using a 10 Mbits/sec communication link are very close to those obtained on the dedicated hardware (low communication overhead) slotted ring architecture. In addition, speedup results obtained are comparable to those published for a shared memory architecture [20,70].

II **Standard Circuit Simulation and Decomposition Methods**

This chapter explores how different algorithms solve circuit simulation at the electrical level. Section 2.1 explains in some detail the steps needed in the simulation using Direct Methods (DM), and which steps represent the bottleneck in the simulation. Sections 2.2 and 2.3 present different decomposition methods and explain how to accelerate the solution process by avoiding the solution of a large matrix, and by exploiting waveform properties such as latency and multirate integration algorithms.

2.1 Advantages and Disadvantages of Standard Circuit Simulation

Circuit simulation in the time domain involves the solution of a system of nonlinear ordinary differential equations (ODE) of the form

$$\dot{q}(v(t)) = -f(v(t), u(t)), v(0) = V, t \in [0, T] \quad (\text{EQ 1})$$

where q is the sum of the charges due to the capacitors connected to each node, f is the sum of the currents charging the capacitors at each node, $v(t) \in \mathbb{R}^n$ is the set of unknown node voltages and $u(t)$ is the set of input source voltages.

The standard approach to solving the system of nonlinear ODEs is commonly referred to as direct methods (DM) and is used in the SPICE2 simulator. This method involves three main parts:

1. First, an implicit stiffly stable integration method is used to discretize the differential equations which describe the system into a set of nonlinear algebraic equations.
2. Second, a modified Newton iteration scheme is applied to convert the set of nonlinear equations into a sequence of linear algebraic equations.
3. The set of linear algebraic equations generated in each Newton iteration is solved by sparse Gauss elimination or LU decomposition.

Appendix A describes the section of these stages in some detail. The main advantage of the DM is that it is very reliable and can be used across a variety of different technologies and element types. A simplified flow diagram for the solution of these equations by a conventional circuit simulator is shown in figure 1. The most computationally intensive part corresponds to the inner loop of this dataflow, which is composed of two phases: the *formulation* phase and the *solution* phase. The time required to form the Jacobian matrix corresponds to the *formulation* phase represented by step 5 and 6, and the time required to solve the system of sparse, linear equations corresponds to the *solution* phase represented by steps 7 and 8.

For small to medium sized circuits containing MOS devices, the model evaluation and load times dominate the total CPU time for the simulation. The first problem in the standard direct method solution is that the sparse linear equation solution time grows faster than linearly with circuit size. The *solution* phase ranges from $O(n^{1.2})$ to $O(n^{1.5})$, where n is the size of the matrix. Although the *solution* phase makes a negligible contribution to the total run time for small circuits, this phase dominates the run time for very large circuit simulation. The crossover between the *solution* and *formulation* phase depends on the complexity of the equations describing the model of the elements in the circuit and the

```

1. Read the circuit description and initialize data structures
While ( $t_{n+1} \leq T_{stop}$ ) {
    2. Update values of independent sources at  $t_{n+1}$ 
    3. Predict values of unknown variables at  $t_{n+1}$ 
    4. Apply integration formulas to linear capacitors and inductors
    While (Newton-Raphson has not converged) {
        5. Apply integration formulas to nonlinear capacitors and inductors
        6. Apply Newton-Raphson method to nonlinear circuit elements
        7. Assemble and LU factorize linear circuit equations
        8. Solve linear circuit equations
    }
    9. Use Local Truncation Error estimation to test accuracy of solution at  $t_{n+1}$ 
    10. Save solution if it is acceptable
    11. Select new time step and new  $t_{n+1}$ 
}
12. Print output data and stop

```

FIGURE 1 Solution of equations by a conventional circuit simulator

size of the circuit. Figure 2 shows an execution profile of a typical VLSI memory circuit with 3700 equations¹.

Therefore, any technique which attempts to reduce overall circuit simulation run times must reduce both the model evaluation time and the linear equation solution time to be effective for circuits containing more than 1000 nodes.

1. Example taken from Paul Cox et al in "Direct Circuit Simulation Algorithms for Parallel Processing", IEEE Trans. on CAD, Vol. 10, No. 6, June 1991.

FIGURE 2

SPICE execution profile of a VLSI memory circuit with 3700 equations

The second disadvantage of the standard circuit simulation technique is that certain waveform properties, such as latency and multirate behavior, are difficult to exploit since the direct application of the integration method forces every differential equation in the system to be discretized identically. This discretization must be fine enough so that the fastest-changing circuit variable in the system is accurately represented. A multirate algorithm will pick different discretization points, or time steps, for each differential equation in the system, ensuring that each can use the largest time step that would accurately reflect the behavior of its associated variable. This property is typical for MOS digital circuits which inherently exhibit this kind of behavior because of different transistor sizes and different capacitance values at each node in the circuit.

Latency is another waveform property which refers to a situation where the value of a variable x does not change with respect to time; latency exploitation involves computing the waveform only at time points when the signal is changing. Latency is viewed as a subset of the multirate behavior. Since the node voltages of most digital circuits do not

vary (latent or inactive) for most of the simulation interval, exploiting latency using event-driven techniques can be effective in the solution of non-linear transient analysis.

Exploiting either property at the circuit level can greatly reduce the number of time points computed for each waveform. In fact, the SPICE program exploits latency by using a **bypass** scheme. In this technique, each device is checked to see if any of its terminal voltages and branch currents of a circuit element changed significantly from the previous Newton-Raphson iteration; their contributions to the Jacobian and the Newton residue are not re-evaluated, and the values computed during the previous iteration are used.

The evolution of non-standard circuit simulators in the last decade makes use of these waveform properties, exploiting circuit regularity, latency and multirate properties. This new generation of simulators requires some form of system decomposition technique which can be applied to any of the three levels of the standard simulation approach, linear, non-linear and differential equation levels. There are two forms of departure from the standard circuit simulation. These are the **tearing methods**, which retain the convergence and stability properties of the standard approach, and the **relaxation methods** which are characterized by completely different convergence and stability properties. These methods are presented next.

2.2 Tearing Methods

The tearing approach can be referred to as a system decomposition of non-linear circuit level or linear circuit level. In both cases the Jacobian matrix A is transformed in a block structured matrix type of Bordered Block Diagonal (BBD) or Bordered Block Triangular (BBT) as shown in figure 3, in which each small block in either BBD or BBT structure represents a small subnetwork of the entire circuit. To partition a circuit into small subcircuits, one must decide which variables in the functional blocks are “endogenous”

FIGURE 3

BBD and BBT matrix structures

(interact only inside the block); and which are “exogenous” (interact with the variables at the external level).

This choice characterizes the type of decomposition actually employed. For example, if one chooses voltages as exogenous variables, the Node Tearing approach results. On the other hand, if the exogenous variables are branch currents, Branch Tearing is selected. In both cases, exploitation of latency of the subnetworks allows for a reduction in computation for those subnetworks which are inactive. Appendix B describes both tearing methods.

These special matrix structures immediately suggest how the equations may be solved in several small uncoupled blocks and then combined with the intermediate results to obtain the final solutions.

In both tearing methods, the decomposition is performed inside the Newton iteration at each simulation time point. This decomposition of a large circuit in small subcircuits is a modular approach found in most third generation simulators. IC simulators such as SLATE[50] and MACRO[78] employ this technique. As well, this partitioning allows

advantage to be taken of machines with parallel architectures. Hardware resources to accelerate the simulation will be addressed in the next chapter. A new set of iterative algorithms is discussed next, based on “temporal” decomposition methods.

2.3 Relaxation Based Methods

The relaxation methods also make use of partitioning but it is performed before the linearization. In fact, relaxation methods can be applied at different stages of (EQ 1) in a number of ways, as illustrated in figure 4, which shows three alternatives for solving non-linear differential equations. In a relaxation method, the unknowns are decoupled, solving for one unknown at a time and leaving the others fixed. The term relaxation comes from

FIGURE 4

Solution of equations with use of relaxation methods

“relaxing” one variable at a time, holding the others constant. Whatever the level at which relaxation is applied, the two most common relaxation techniques are the Gauss-Jacobi method and the Gauss-Seidel method [46].

Although relaxation algorithms have been applied to both bipolar and MOS circuits, the emphasis is placed on the simulation of MOS digital circuits because individual elements are already decoupled to a large extent.

2.3.1 Linear Relaxation

For the solution of the linear equation $Ax = b$, relaxation methods can replace direct methods for the solution of (EQ A.6) by splitting A into $L + D + U$, where $L \in \mathfrak{R}^n$ is strictly lower triangular, $D \in \mathfrak{R}^n$ is diagonal, and $U \in \mathfrak{R}^n$ is strictly upper triangular.

For the Gauss-Jacobi technique (G-J), the following form is applied to the solution of linear system of equations as given by (EQ A.6).

Gauss-Jacobi

$$Dx^{k+1} = -(L + U)x^k + b \quad (\text{EQ 2})$$

$$\text{or } x^{k+1} = -D^{-1} ((L + U)x^k - b) \equiv M_{GJ}x^k + D^{-1}b \quad (\text{EQ 3})$$

An interesting feature of the G-J technique is that the order in which the separate variables are solved can be arbitrary. This is not true for the Gauss-Seidel technique (G-S) as defined below.

Gauss-Seidel

$$(L + D)x^{k+1} = -Ux^k + b \quad (\text{EQ 4})$$

$$x^{k+1} = -(L + D)^{-1} (Ux^k - b) \equiv M_{GS}x^k + (L + D)^{-1}b \quad (\text{EQ 5})$$

where x^k and x^{k+1} are the values of x at the k and $k+1$ iteration respectively.

Notice that in the G-J method each x_i^{k+1} is computed using the iteration values $x_j^k, j = 1, \dots, n$, the values from the previous iteration. In the G-S method, the latest iteration values are used as soon as they become available. Therefore, in the G-J method the computation of all $x_i^{k+1}, i = 1, \dots, n$ can proceed in parallel.

Both schemes require an initial guess $[x^{(0)}] = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]$. If we assume that the diagonal components are strong, and dominant, then the value of the i th variable is mainly governed by the i th equation. It is well known that a necessary and sufficient condition for the iterations defined by (EQ 3) and (EQ 5) to converge in the solution of (EQ A.6), independent of the initial guess $[x^{(0)}]$, is that the eigenvalues of M_{GJ} and M_{GS} be inside the unit circle in the complex plane [67]. From a computational point of view, a more efficient approach is to check if A is a strictly diagonally dominant matrix.

$$\sum_{j=1}^n |a_{ij}| < |a_{ii}| \text{ for } (1 \leq i \leq n) \quad (\text{EQ 6})$$

Of course, the more dominant the diagonal, the more rapid is the convergence to the solution.

In general, the G-S technique is preferred over the G-J because it converges faster. In the case of a lower triangular matrix A , the G-S converges in one iteration while the G-J takes n iterations. Therefore, in the G-S scheme, the order of computation is important and could affect the time for the solution drastically. As an extreme case, if A is lower triangular and one solves in the order $x_n, x_{n-1}, \dots, x_1$, the solution takes n iterations instead of one. Often a knowledge of the problem may help in the ordering. If the response to an input change is being studied, the order would naturally be the one in which various node voltages occur as the signal flow path is traced.

The computational cost of both the G-J and G-S methods is $O(n)$, compared with $O(n^{1.2-1.5})$ for direct, sparse matrix techniques. Thus relaxation methods are only better from a computational point of view if the number of iterations needed to obtain the convergence is of the order of $n^{0.1}$.

[40] and [52] found that it is more effective to use relaxation techniques at an earlier stage in the solution process so that the waveform properties can be exploited easily.

2.3.2 Non-Linear Relaxation Methods

The basic idea of relaxation can also be applied to solve a system of non-linear equations of the form:

$$f_1(x_1, x_2, \dots, x_n) = 0$$

$$f_2(x_1, x_2, \dots, x_n) = 0$$

$$\vdots$$

$$f_n(x_1, x_2, \dots, x_n) = 0$$

These can be decoupled at the non-linear equation level using the G-S or G-J techniques. Let the initial guess be $[x^{(0)}] = [x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}]$. To get $x_i^{(1)}$ using the G-J we solve

$$f_i(x_1^{(0)}, x_2^{(0)}, \dots, x_i^{(1)}, \dots, x_n^{(0)}) = 0 \quad \text{for } x_i^{(1)}.$$

Using the G-S we solve

$$f_i(x_1^{(1)}, x_2^{(1)}, \dots, x_{i-1}^{(1)}, x_i^{(1)}, x_{i+1}^{(0)}, \dots, x_n^{(0)}) = 0 \quad \text{for } x_i^{(1)}$$

In either case we are solving non-linear equations in just one unknown.

Let us assume that we decouple the unknowns using the G-S and then solve the non-linear equations for one unknown using Newton-Raphson. We therefore have two levels of iterations; an outer Seidel iteration and an inner Newton iteration. This method of solving non-linear equations is referred as the Seidel-Newton method. If the G-J is used in the outer loop instead, it is named the Jacobi-Newton method. For the Seidel-Newton method, in every Seidel iteration we solve n non-linear equations which are decoupled. Each of these equations is solved in many Newton-Raphson iterations. In principle, both inner and outer loop iterations must be carried to convergence. However, it is possible to trade-off some Newton iterations for Seidel iterations. One could, for example, do just one Newton iteration but carry the outer Seidel iterations to convergence. This is the method applied by the SPLICE1.7 program [52], using single node decomposition. This method, called Iterative Timing Analysis (ITA), is an incremental method, i.e. the voltages of the circuit nodes are determined at each time point before starting the computation at the next time point. Event-driven analysis and selective trace are used to exploit the temporal sparsity of the electrical network. The SPLICE1.7 is an order of magnitude faster for the simulation of MOS digital circuits, but for analog circuits it is slower because of convergence problems, when strong coupling exist between nodes.

The diagonal dominance requirement for G-S iteration must be satisfied to guarantee the convergence of the Seidel-Newton iteration. In circuit terms, this requirement can always be met by placing a grounded capacitor at every node and choosing an appropriate step size. Grounded capacitors appear as $\frac{C}{h}$ terms in the diagonal position of the conductance matrix G . Therefore, h (simulation step size), can be reduced until the $\frac{C}{h}$ terms are greater than the sum of all off-diagonal terms. Convergence problems exist where there is a strong coupling between two nodes. For example, a floating capacitor generates $\frac{C}{h}$ terms in the diagonal and off-diagonal position. Therefore, reducing the step size is not as effective, especially when the floating capacitor is much greater than the grounded capacitors at

each node. Thus node by node decomposition is not an effective way to solve using relaxation methods. This problem of circuit partitioning will be addressed in Chapter IV.

As in the linear case, the Gauss-Seidel method tends to converge faster than Gauss-Jacobi. Reordering of the equations affects the speed of the Gauss-Seidel method. Let us call P the dependency matrix which represents the zero-non-zero structure of the Jacobian matrix from (EQ A.5). Then if P is lower triangular, only one iteration of the outer Gauss-Seidel relaxation loop is needed, provided that the inner Newton-Raphson loop is run to convergence. In fact, it has been proven that if only one Newton iteration is taken, then the convergence rate of the Seidel-Newton is linear. The use of relaxation at the non-linear equation level coupled with the single Newton-Raphson inner loop, only needs the computation of the partial derivative of f_i with respect to x_i , resulting in a considerable saving of computer time per iteration. This technique has been successfully applied in the simulation of MOS digital circuits because of the quasi-unidirectionality of the MOS devices from gate to source and gate to drain. A key to applying relaxation techniques to the solution of circuit equations is the reordering of the equations so that P is almost lower triangular. Thus the convergence speed of non-linear relaxation depends primarily on the degree of coupling between the equations in the system and the order in which they are processed.

The concepts of G-S and G-J iteration methods can be extended to the non-linear differential equations has shown in the top level of relaxation techniques in figure 4. This method has been named Waveform Relaxation (WR).

2.3.3 Waveform Relaxation Method

Waveform Relaxation was proposed in 1982 [35] with the introduction of the RELAX program. The method has been shown to allow for significant circuit simulation

time reduction [35,61]. The WR method is an iterative method based on a relaxation scheme which applies decomposition directly on the system of differential equations.

If we consider the following system of ODE's:

$$f_1(\dot{x}(t), x(t), u(t)) = 0$$

$$f_2(\dot{x}(t), x(t), u(t)) = 0$$

$$\vdots$$

$$f_n(\dot{x}(t), x(t), u(t)) = 0$$

with $x(0) = x_0$ and where $x(t)$ is the vector of unknown variables at time t , $\dot{x}(t)$ is the vector of the time derivatives of $x(t)$ and $u(t)$ is the vector of input voltages. This set of equations has to be solved in a time interval $[0, T]$.

The basic idea of the G-S waveform relaxation algorithm can be illustrated as follows:

1. Set the iteration number $k = 0$ and make an initial guess of the waveforms: $x^0(t)$ for $t \in [0, T]$ such that $x^0(0) = x_0$.

2. For each $i = 1, 2, \dots, n$ solve for $x_i^k(t)$; $t \in [0, T]$ from the following equation:

$$f_i(\dot{x}_1^k, \dots, \dot{x}_i^k, \dot{x}_{i+1}^{k-1}, \dots, \dot{x}_n^{k-1}, x_1^k, \dots, x_i^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u) = 0, x_i^k(0) = x_0 \quad (\text{EQ 7})$$

3. $k = k + 1$ and repeat process 2 until the difference between $x^k(t)$ and $x^{k-1}(t)$ is sufficiently small over the simulation interval $[0, T]$, i.e., $\max_{t \in [0, T]} |x^k(t) - x^{k-1}(t)| \leq \epsilon$.

Note that differential equation (EQ 7) contains only one unknown variable x_i^k . The variables $x_{i+1}^{k-1}, \dots, x_n^{k-1}$ are known from the previous iteration and the variables $x_1^k, \dots, x_{i-1}^k$ have already been computed in the current iteration.

In the Gauss-Jacobi WR method the decoupled differential equation is:

$$f_i(\dot{x}_1^{k-1}, \dots, \dot{x}_i^k, \dot{x}_{i+1}^{k-1}, \dots, \dot{x}_n^{k-1}, x_1^{k-1}, \dots, x_i^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u) = 0, \quad v_i^k(0) = v_{i0} \quad (\text{EQ 8})$$

The unknown variable is again $\dot{x}_i^k$, but all other variables are taken from the previous WR iteration $k-1$.

In this decoupling at the differential equation level, each decoupled differential equation is independently analyzed through the overall simulation interval. The main feature of the WR method is its easy exploitation of the quasi-unidirectional property of MOS circuits and the multi-rate nature of large scale integrated circuits.

WR algorithms applied to circuits can be formulated for systems of differential equations of the form of (EQ 1). If the Gauss-Seidel WR method is applied, the decoupled differential equation (EQ 7) that has to be solved iteratively becomes:

$$\sum_{j=1}^i C_{ij}(\dot{v}_1^k, \dots, \dot{v}_i^k, \dot{v}_{i+1}^{k-1}, \dots, \dot{v}_n^{k-1}, u) \dot{v}_i^k + \sum_{j=i+1}^n C_{ij}(\dot{v}_1^k, \dots, \dot{v}_i^k, \dot{v}_{i+1}^{k-1}, \dots, \dot{v}_n^{k-1}, u) \dot{v}_i^{k-1} = f_i(\dot{v}_1^k, \dots, \dot{v}_i^k, \dot{v}_{i+1}^{k-1}, \dots, \dot{v}_n^{k-1}, u) \quad (\text{EQ 9})$$

for $v_i(t) : t \in [0, T]$ with the initial condition $v_i^k(0) = v_i$.

The variable $\dot{v}_i^k$ is the only unknown. The initial waveform for the node voltage $v_i^0(t)$ has to be guessed. These equations are solved until convergence of the computed waveforms, i.e., until $\max_{t \in [0, T]} |v_i^k(t) - v_i^{k-1}(t)| \leq \epsilon$.

The application of the WR method is better illustrated with an example taken from [35]. In this example, the one-bit dynamic shift register cell shown in figure 5a and figure 5b is partitioned as shown in figure 6. Subcircuit S_1 is used to compute $v_1(t)$. Other

FIGURE 5

(a) Logic representation of a dynamic shift register

(b) Circuit representation of (a)

voltages which can influence v_1 are included in the subcircuit S_1 . $v_1^k(t)$ represent the k th iteration for $v_1(t)$. A Gauss-Seidel process has been assumed in the ordering solution. When calculating $v_1^k(t)$, $v_2^{k-1}(t)$ can be treated as an independent source like $u_1(t)$ and $u_2(t)$. Once the entire waveform for $v_1^k(t)$ has been found, the subcircuit S_2 is used to find $v_2(t)$. In computing waveform $v_2^k(t)$, $v_1^k(t)$ which has already been computed from S_1 and $v_1^{k-1}(t)$ which is the previous waveform computed from subcircuit S_1 , are treated as independent sources.

The relevant decoupled differential equations corresponding to S_1 , S_2 and S_3 are:

$$(C_1 + C_2 + C_3) \dot{v}_1^k - i_1(v_1^k) + i_2(v_1^k, u_1) + i_3(v_1^k, u_2, v_2^{k-1}) - C_1 \dot{u}_1 - C_3 \dot{u}_2 = 0$$

$$(C_4 + C_5 + C_6) \dot{v}_2^k - C_6 \dot{v}_1^{k-1} - i_3(v_1^k, u_2, v_2^k) - C_4 \dot{u}_2 = 0$$

$$(C_6 + C_7) \dot{v}_3^k - C_6 \dot{v}_2^k - i_4(v_3^k) + i_5(v_1^k, v_2^k) = 0$$

FIGURE 6 Circuit decomposition from figure 5 using waveform relaxation

These equations correspond to the k th iteration. For guaranteed convergence of these equations independent of the initial guess, it is necessary to have a ground capacitor at each node, such as C_2 , C_5 and C_7 . These mild conditions will make the matrix $C(v, u)$

in (EQ 9) a strictly diagonally dominant uniformly in v, u . These conditions are satisfied for most MOS circuits. Next, a comparison between ITA and WR techniques is made, presenting the improvements of these algorithms through the evolution of the SPLICE and RELAX simulators respectively

2.3.4 ITA and WR Algorithm Speedups

For a good comparison between ITA and WR refer to the graph in figure 7. This graph is related to the solution of the circuit in figure 5, where u_1 and u_2 are the input voltages to the circuit, known for the entire simulation interval, v_1 , v_2 and v_3 are the respective node voltages. In ITA the order of computation is $v_1(t_1), v_2(t_1), v_3(t_1)$ until all voltages converge. The time step size is then incremented, and a new set of voltages are computed at time t_2 . The computation is thus column wise, though, in the WR the order of computation is row wise. The entire waveform $v_1(t)$ until t_n is computed first, then computing $v_2(t)$ and $v_3(t)$ proceeds successively for the same interval of time, followed by iteration in the same order until the waveforms converge.

FIGURE 7

Graphical comparison in the Gauss-Seidel iteration
loop between WR and ITA

A prototype version of ITA using Backward-Euler integration, and node by node decomposition with a fixed time step was implemented in SPLICE1.7. As mentioned earlier, the biggest problem with this algorithm is that when the circuit is a tightly coupled analog circuit with large forward gain and capacitive feedback like NMOS operational amplifiers, the simulation time takes more than twice the CPU-time used by SPICE2 [40]. The main reason for this is that the convergence is very slow in the presence of the large floating capacitors and small drain and source resistors usually found in the detailed MOS transistor model. The evolution of ITA took these problems in consideration by identifying strongly-coupled components in the circuit automatically and grouping them to form subcircuits. This process, referred as **circuit partitioning**, has been implemented in iSPLICE3.1. Since the variables associated with the subcircuits are assumed to be tightly-coupled, each subcircuit can be solved using direct method techniques, and the relaxation method can be applied between subcircuits. This partitioning technique, inherited from the RELAX2 program, will be described in more detail in Chapter IV.

Another characteristic of iSPLICE3.1 is that it uses a global variable time step algorithm in which the components in the system are integrated using a single common time step. This time step is selected based on the fastest changing variable in the system, the same strategy used in direct methods. However, the selection is based only on the active subcircuits, which are identified using a selective trace algorithm.

When the active subcircuits converge at a time point, t_i , the local truncation errors for their internal variables are estimated [3] and the new global time-step, h_{next} , is set to the smallest recommended step in the system, h_{min} . If the accuracy in the solution computed at t_i is unacceptable, the solution is rejected and the integration is retried with a smaller time step. Similarly, if the iterations do not converge within a specified number of iterations, the time step is rejected and a smaller step is used. The ITA method in the

iSPICE3.1 simulator has proven to be very robust and accurate, with a 5 to 10 times faster rate than SPICE2.

Parallel to the evolution of the ITA techniques, waveform relaxation has also evolved in the program environment RELAX.

WR has similar problems when used in node by node decomposition. It needs many waveform iterations to converge to the right solution when the nodes are strongly coupled. Therefore a block relaxation method was adopted with the same decomposition method used in ITA. This is basically the first step in almost every WR based program. Partitioning well is a difficult task for several reasons. When so many nodes are lumped together, the advantages of using relaxation can be lost, but if any tightly coupled nodes are not lumped together, the WR algorithm will converge very slowly.

For faster execution, waveform relaxation simulators can also check for latency in a subcircuit. A scheme to trace the signal flow path should be incorporated so that dormant segments are not analyzed. For most digital circuits, the latency intervals of a subcircuit usually cover a large portion of the entire simulation time interval $[0, T]$. Hence the implementation of this technique can provide a considerable saving of computing time.

Another technique to obtain speed in the simulation with WR is based on the partial convergence of a waveform (PWC) during the previous two iterations. Often some pieces of the circuit will converge much more rapidly than others. The overall efficiency of the WR can be improved if the waveforms that have already converged are not recomputed on every subsequent iteration (third, fourth, etc.) as shown in figure 8.

A new technique was exploited in RELAX2 [72], which is a modified WR algorithm that uses window techniques. Digital circuits can be broken up into two very broad classes: circuits with logic feedback loops (finite state machines, asynchronous circuits,

FIGURE 8

Partial waveform convergence with two iterations

digital oscillators) and circuits without logic feedback loops (combinational circuits, PLA). RELAX2 shows that most MOS digital circuits without a logic feedback loop converge in less than ten iterations. However, circuits with a logic feedback loop may take many more iterations to converge and the number of iterations required is proportional to the length of the simulation interval. Therefore, instead of computing the transient behavior of each subcircuit for the entire simulation interval, the relaxation is only performed for a small piece of the user-defined simulation interval of time. Exactly how large a piece of the waveform, referred to as a **window**, to use is determined automatically at the beginning of every WR iteration. If the simulation interval is divided into smaller windows $[0, T_1], [T_1, T_2], \dots, [T_{n-1}, T_n]$, then WR is applied only in the first window, $[0, T_1]$, until the waveforms converge. Then a second window, $[T_1, T_2]$, is selected and WR is applied

within this interval until the waveforms converge. This continues until the entire simulation interval is covered. Note that the WR method converges more rapidly as the window size is made smaller. Its drawback is that it constrains the time step to the window size, which can lead to additional time point computation and a less efficient latency exploitation. The scheduling overhead also increases when the windows become smaller, because each subcircuit must be scheduled for each window. A trade-off exists between the potential gain due to windowing and the overhead associated with it.

The advantage of breaking the interval of simulation into windows is the decrease of memory usage. In the WR method, the whole waveform of each node has to be stored in memory during the iteration process. If windowing is applied, only the part of the waveform that is in the current window has to be stored.

If the window size is reduced to a single time step, the method becomes a one-step WR, an incremental technique similar to ITA. Results of Saleh's research [54] show that is better to exploit the multi-rate properties of circuits rather than simply the latency property as it offers a potentially larger speed improvement.

2.4 Conclusions

This chapter presented three different simulation algorithms. The first algorithm presented uses the Direct Methods, which have become the standard technique in modern simulators, mainly because the approach is reliable and can be used across a variety of technologies and element types. The major constraint is imposed by the numerical techniques on the step sizes used in the integration process. The second largest constraint is that for very large circuits, in the Newton-Raphson phase, the linear equations are solved using a DM approach such as LU decomposition. The solution time for these matrices grows superlinearly with respect to the matrix order size. In trying to avoid solving these

large matrices, new circuit decomposition methods have emerged in the last ten years which can effectively reduce the computation time.

Two decomposition methods were also presented: tearing decomposition and temporal decomposition. These methods differ in the way feedback between the blocks of the decomposition is treated. Tearing methods take the feedback fully into account, while temporal decomposition methods achieve decomposition by cutting feedback paths and then performing an inexpensive simulation which approximates the effects that feedback has on the blocks of the decomposition.

From among the temporal decomposition methods, two techniques were discussed: the ITA method which decomposes the circuit at the non-linear equation level, and the WR method which decomposes the circuit at the differential equation level. Which method to use depends primarily on the coupling effect between subcircuits. Gallivan [17] shows that there is no clear winner for the benchmark circuits, but if the circuit is very tightly coupled, then DM should be used. If the circuit is large and loosely coupled, WR should be used; if the circuit is large and moderately coupled, as is usually the case, ITA should be used. Although significant speedups have been reported using relaxation methods over DM, there is still need to increase the simulation speedup for circuits of today increasing complexity.

In the next chapter, hardware resources are presented which can increase the speedup of the simulation by exploiting the algorithm parallelism. WR and ITA algorithms are also presented in the parallel environment to obtain maximum efficiency in the electrical circuit simulation level.

III Parallel Architectures and Algorithms in the Acceleration of Circuit Simulation

Relaxation methods have provided an increase in the speed of simulation with up to two orders of magnitude for large MOS circuits, compared to standard circuit simulation. However, these improvements may not be enough for the largest problems. With today's technology million of transistors are being packed in a chip, and simulation at the circuit level with these densities is considered prohibited. Therefore, dedicated hardware resources are necessary to reduce the run times for the simulation of VLSI systems. Different computer architectures have been used in the acceleration of these algorithms, which are presented in section 3.1, based on Flynn's taxonomy. A ring architecture implemented at Carleton University which has been selected to be used as one of the testbeds for the distributed WR algorithms is presented in section 3.2. A performance study using this ring architecture for an iterative matrix solver algorithm using 4 processor elements is included. Next, a literature review of different circuit simulation algorithms mapped into different parallel machines is presented. Section 3.3 concentrates on parallelism for the DM while section 3.4 is dedicated to parallelism as related to Relaxation Methods. Finally, section 3.5 introduces the framework of this dissertation based on the exploitation of WR in a distributed circuit simulation environment. Sun workstations are also considered for use as a vehicle to test our algorithms based on the current proliferation of these multicomputers at major universities and research centers. Their expendability and the

generality of these architectures make them very suitable in the distributed framework for circuit simulation.

3.1 Parallel Processing System Classification

The most widely accepted classification of parallel processing systems is the Flynn classification [15]. Flynn's taxonomy classifies architectures on the presence of single or multiple streams of instructions and data. This yields the four categories below:

SISD (Single instruction, single data stream) defines the classical Von Newmann machine architecture computer with practically no parallelism. However, it may also represent some more sophisticated systems, where certain methods of parallelism have been implemented, such as multiple functional units (IBM 360/91, CDC 6600, CYBER 205) or pipelining (VAX 8600) or both.

MISD (Multiple instruction, single data stream), whose architecture would involve multiple processors applying different instructions to a single datum; this hypothetical possibility is generally deemed impractical.

SIMD (Single instruction, multiple data streams) involves multiple processors simultaneously executing the same instruction on different data.

MIMD (Multiple instruction, multiple data streams) involves multiple processors autonomously executing diverse instructions on diverse data.

SIMD architectures typically employ a central control unit, multiple processors, and an interconnection network (IN) for either processor-to-processor or processor-to-memory communications (see figure 9). The control unit broadcasts a single instruction to all processors, which execute the instruction in lockstep fashion on local data. The

FIGURE 9

SIMD execution

interconnection network allows instruction results calculated at one processor for use as operands in a subsequent instruction. Individual processors may be allowed to disable the current instruction. A system of this type is also called an array processor, such as the ILLIAC-IV, BSP (Burroughs Scientific Processor). One variant of processor array architectures involves using a large number of one-bit processor. In bit-plane architectures, the array of processors is arranged in a symmetrical grid (such as 64x64) and associated with multiple “planes” of memory bits that correspond to the dimensions of the processor grid (see figure 9). Processor n (P_n), situated in the processor grid at location (x, y) , operates on the memory bits at location (x, y) in all the associated memory planes. Usually operations are provided to copy, mask, and perform arithmetic operations on entire memory planes, as well as on columns and rows within a plane. Thinking Machines' Connection Machine organizes as many as 65,536 one-bit processors as sets of four-processor meshes united in a hypercube topology.

MIMD architectures employ multiple processors that can execute independent instruction streams using local data. Thus, MIMD computers support parallel solutions

FIGURE 10 Bit-plane array processing

that require processors to operate in a large autonomous manner. Although software processes executing on MIMD architectures are synchronized by passing messages through an interconnection network or by accessing data in shared memory units, MIMD architectures are asynchronous computers, characterized by decentralized hardware control. MIMD computers support higher level parallelism that can be exploited by “divide and conquer” algorithms organized as largely independent sub-calculations.

The MIMD systems can be partitioned further based on the underlying architecture. There are two main groups: shared memory systems and distributed memory systems.

3.1.1 Shared-Memory Systems

Shared memory architectures accomplish interprocessor coordination by providing a global, shared memory that each processor can address, as in figure 11.

Coordinating processors with shared variables requires an atomic synchronizing mechanism to prevent one process from accessing a datum before another finishes updating it.

Typically, each processor in a shared memory architecture also has a local memory used as a cache. Multiple copies of the same shared memory data, therefore, may exist in various processors' caches at a given time. Maintaining a consistent version of such data is the cache coherency problem, which requires providing new versions of the cache data to each involved processor whenever a processor updates its copy.

FIGURE 11**Shared memory configuration**

The communication network in a multiprocessor is of outmost importance. When a program is distributed among a number of processors for execution, there is always a need for transmission of intermediate results between the tasks which are scheduled to run on different processors. In fact, some processes will not be able to proceed without receiving the results from other processes. This calls for synchronization between different processes belonging to the same program. The communication network plays a crucial part in all of these events since it has to efficiently transmit all of the data, intermediate results, scheduling, and synchronization control signals.

The need for scheduling, synchronization, and communication in a multiprocessor system introduces an overhead in the system operation. Scheduling and synchronization depend, of course, on the algorithms used to implement them. However, their ultimate efficiency strongly depends on the quality of the communication system. It thus makes sense to regard the overhead of a multiprocessor system as the communication overhead.

The throughput of a multiprocessor certainly depends on the number of processors and the quality of each individual processor configured into the system. However, an effort to improve the throughput of the processors, without paying attention to the efficiency of the communication system, may prove counter-productive. If the communication overhead is too high, a single processor may handle the problem faster than a number of processors working in parallel [64].

Figure 12 illustrates some major alternatives for connecting multiple processors to shared memory. These include:

Bus Interconnection: Time-shared buses offer a fairly simple way to give multiple processors access to a shared memory. A single, time-shared bus effectively accommodates a moderate number of processors (from 4 to 20), since only one processor accesses the bus at a given time. See figure 12a. This bus system is implemented in the Sequent series multiprocessor system. A single-bus system offers great configuration flexibility to both the user and the designer, since components and subsystems can be easily added to or disconnected from the bus. Unfortunately, this type of system suffers from serious drawbacks, the most serious being bus bottleneck. Only two devices at a time can be used to establish communications through the system bus. If one of the processors fails, the multiprocessor can continue to operate with decreased computing power. Though all modern multiprocessors are designed for "graceful degradation" capabilities, a bus failure is catastrophic in a single-bus architecture.

FIGURE 12

MIMD shared-memory interconnection schemes:
 (a) single bus interconnection (b) multiple bus system
 (c) crossbar switch network (d) 8x8 omega MIN routing
 a P3 request to M3

These serious disadvantages of a single-bus system can be alleviated if multiple buses are implemented. Certainly, the failure of a single bus does not have to be catastrophic. If we have b buses, then up to b simultaneous interconnections can be achieved, alleviating the bottleneck problem. Of course, other problems arise. Looking at figure 12b, one can see that multi-port capability requires complicated and expensive extra logic in each device. Therefore, the number b can not be too high. A dual-bus system $b = 2$ can be achieved at a reasonable cost. For instance, the Alliant multiprocessor system has a dual system bus, one between its cache and main memory, and the other bus connecting the processors.

Crossbar Interconnection: Crossbar interconnection technology uses a crossbar switch of n^2 cross-points to connect n processors to n memories (see figure 12c). Information is routed through Cross-point Switches (CS), which contain multiplexing and arbitration logic networks. If more than one processor attempts to access the same memory module, a priority queue can be established by the arbitration network. The main advantage of a crossbar network is its potential for high throughput by multiple, concurrent communication paths. Its main disadvantage is its exceedingly high cost and complex logic. The Alliant FX/8 is a commercial architecture that uses a crossbar scheme to connect processors and cache memories.

Multistage Interconnection Networks: Multistage interconnection networks (MINs) strike a compromise between the price/performance alternatives offered by crossbars and buses. An $N \times N$ MIN connect N processors to N memories by deploying multiple “stages” or banks of switches in the interconnection network pathway.

When N is a power of 2, one approach is to employ $\log_2 N$ stages of $N/2$ switches, using 2×2 switches. A processor making a memory access request specifies the desired destination (and pathway) by issuing a bit-value that contains a control bit for each stage.

The switch at stage i examines the i th bit to determine whether the input (request) is to be connected to the upper or lower output.

Figure 12d shows an omega network connecting eight processors and memories, where a control bit equal to zero indicates a connection to the upper output. Expendability is a significant feature of such a MIN, since its communication diameter is proportional to $\log_2 N$. The BBN Butterfly, for example, can be configured with as many as 256 processors.

3.1.2 Distributed Memory Systems

Distributed memory architectures connect processing nodes (consisting of an autonomous processor and its local memory) with a processor-to-processor interconnection network (see figure 13). Nodes share data by explicitly passing messages through the interconnection network, since there is no shared memory. These architectures have prin-

FIGURE 13

Distributed memory configuration

cipally been constructed in an effort to provide a multiprocessor architecture that will accommodate a significant increase of processors and will satisfy the performance requirement of large scientific applications characterized by local data references.

Various interconnection network topologies have been proposed to support architectural expendability and provide efficient performance for parallel programs with differing interprocessor communication patterns. Figure 14 depicts these different topologies. The Hypercube topology architecture has been strongly influenced by the desire to develop a “scalable” architecture. Boolean n -cube or “hypercube” topology uses $N=2^n$ processors arranged in an n -dimensional cube, where each node has $n = \log_2 N$ bidirectional links to adjacent nodes. Individual nodes are uniquely identified by n -bit numeric values ranging from 0 to $N-1$ and assigned in a manner that ensures the values of adjacent nodes differ by a single bit. The communication diameter of such hypercube topology architecture is $n = \log_2 N$.

FIGURE 14

MIMD interconnection network topologies:
 (a) ring (b) mesh (c) tree (d) hypercube

FIGURE 15 The hypercube topology

A uniprocessor, represented by a single node, can be regarded as a zero-cube. Two nodes (processors), interconnected by a single path, form a one-cube. Four nodes, interconnected as a square, form a two-cube: eight nodes, interconnected as a cube form a three-cube, and so on (see figure 15). Commercial hypercubes include the Cosmic Cube, Ametek Series 2010, Intel iPSC, and Ncube/10.

3.2 A Multiprocessor System Interconnected in a Ring Topology

A ring architecture was developed at Carleton University [79] to evaluate DSP algorithms at the behavioral level. This architecture selection was based on the fact that the DSP algorithm data transfer is very deterministic: most of the time, producers and consumers are known and unidirectional. As well, the transfer in a multiple Processor Element (PE) system observes the principle of locality. i.e., most transfers occur between those PEs which are physically closest when the PEs are arranged linearly. The main

disadvantage is the latency introduced by processor intervention when data is passed between non-adjacent PEs. The processor must check the incoming data to determine if it is for itself, and if not then pass it on (data routing). This process consumes valuable processor computational resources that are required if the algorithm simulations are to be performed at or near real-time. The latency introduced can be especially detrimental if overall feedback loops are present in the application. Thus an I/O processor must be added to off-load the data intercommunication routing task from the main processor. Ideally the I/O processor would allow other PEs to be mapped in the address space of the main processor, so that writing to them appears and takes the same amount of time as writing to local memory.

To solve this problem to a great extent, this system utilizes a variant of the ring architecture, coined as the “slotted-ring”. Essentially a separate Input-Output Processor (IOP) is introduced to handle intercommunication between PEs in a pipelined “registered” ring structure. Because the bus is quite fast (currently 35 nsec per time slot), the maximum latency can be kept small with a small number of PEs (≤ 16). Appendix C describes the communication bus of this ring architecture and its performance.

The performance of the Slotted-Ring architecture with 4 processors is tested using an iterative matrix solver for the solution of dense linear equations [80].

3.2.1 Iterative Solution of Dense Linear Equations

A set of linear equations of the form $Ax = b$, where A is an $n \times n$ (known) matrix, and x (unknown) and b (known) are (column) n -vectors is used for the testbed ring architecture. The iterative method used to solve the above system is given by EQ. 10 for the i th element ($i=1,2,\dots,n$), at $k+1$ iteration.

$$x_i^{k+1} = \frac{\left[b_i - \sum_{j \neq i} a_{ij} x_j^k \right]}{a_{ii}} \text{ for } j = 1, 2, \dots, n \quad (\text{EQ 10})$$

The n equations are assumed to be diagonally dominant for the solution to be convergent.

The iteration is started by assigning an initial value to each of the components of x (say, $x_i = 0, i = 1, 2, \dots, n$) and is repeated until the absolute value of the difference for two consecutive approximations stays below a predetermined bound ϵ for all components of x .

Because each PE in the ring multiprocessor system has its own local private memory, the problem is partitioned into tasks whose number correspond to the available number of PEs in the system (N PEs). Therefore, each PE in the ring is assigned a task with two sets of inputs: (i) slices of the matrix A and (ii) corresponding slices of the vector b , consisting of the elements A_{ij} and b_i respectively, where

$$j = 1, 2, \dots, n$$

$$i = (r-1) \times (m+1), \dots, r \times m$$

r : the number of the particular PE ($r = 1, 2, \dots, N$)

m : the number of rows in a slice ($= \frac{n}{N}$)

Assuming that $\frac{n}{N}$ is an integer, the problems consist of a set of N cooperating tasks in which each task communicates with all other tasks. Because all the tasks are of the same size, an even load distribution results for all processors, making the computation and the communication steps for all the tasks the same. Hence any arbitrary assignment of the

problem tasks to the PEs in the ring would result in the same execution time for the problem.

After each iteration, each task computes $R = \|x_i^{k+1} - x_i^k\|$, for all i to check for convergence $R \leq \epsilon$. If convergence has occurred or the maximum allowable number of iterations has been exceeded, the iteration process stops; otherwise x_i^k is replaced by x_i^{k+1} and the iterative step is repeated. In each iteration, each PE broadcasts (distribution of the data to all other PEs) the new computed x_i^{k+1} values and the R_{max} for testing against tolerance.

3.2.2 Performance Study in a Slotted Ring Topology

If the algorithm described above is executed on a uniprocessor, each iteration requires

$$T(1) = n \times (n-1) \times (t_a + t_m) \quad (\text{EQ 11})$$

where $T(1)$ is the iteration time with one PE, t_a and t_m are the times to perform an addition and a multiplication operation respectively.

The performance of this algorithm on a parallel processor depends on both the amount of computation performed by each PE and the intercommunication data transfer between PEs. Therefore, to compute the total iteration time in a multiple processor system we must compute the computation time and the communication time separately to study the performance of the algorithm in the system. The derivations are included in the ring topology in which our test bed multiprocessor system is based.

Iteration Computation Time: Each PE is assigned $\frac{n}{N}$ rows of matrix A and vector b . Since each of the parallel tasks is itself a sequential algorithm, we can determine the complexity of each task independently. Therefore the computation task, t_{comp} , for $\frac{n}{N}$ rows in each PE is:

$$t_{comp} = \left(\frac{n}{N} \right) \times (n-1) \times (t_a + t_m) \quad (\text{EQ 12})$$

Iteration Communication Time: This is the total time to broadcast all x_i^{t+1} values for all the PEs in each iteration. The time for a PE to broadcast a data value to all the processors in a ring topology is t_b . If the transfer of $\frac{n}{N}$ data values is done for each PE, the total communication time is:

$$t_{comm} = n \times t_b \quad (\text{EQ 13})$$

From (EQ 12) and (EQ 13), the iteration time on the ring topology for n PEs, $T(n)$ is given by

$$T(N) = \frac{n}{N} \times (n-1) \times (t_a + t_m) + n \times t_b \quad (\text{EQ 14})$$

From (EQ 11) and (EQ 14) we compute the speedup factor, $S(N)$ as the ratio between these two equations:

$$S(N) = \frac{T(1)}{T(N)} \quad (\text{EQ 15})$$

$$S(n) = \frac{n \times (n-1) \times (t_a + t_m)}{\frac{n}{N} \times (n-1) \times (t_a + t_m) + n \times t_b} \quad (\text{EQ 16})$$

After the derivation of the linear speedup, a parametric study of the algorithm on the ring architecture is undertaken. The results are taken for different values of n , with $N = 4$, $t_b = 200ns$ and $(t_a + t_m) = 2us$. Table 1 shows the speedup as a function of n .

TABLE 1

Speedup obtained in the slotted-ring architecture with 4 PEs for solving n linear iterative equations

n	8	16	20	40	60	80	100	120	140	160	180	200	220	240
$S(N)$	3.5	3.75	3.8	3.9	3.93	3.95	3.96	3.96	3.97	3.97	3.98	3.98	3.98	3.99

FIGURE 16

Iteration times for the theoretical matrix solver (solid-line) versus the actual simulations in the test bed multiprocessor (dotted-lines) for 1 PE and 4 PEs, and the simulation runtimes in a single SUN Sparc-2 station (segmented-line)

The actual simulations on the ring architecture were performed with 4 PEs, whose run times are shown in the figure 16 with dotted lines. The theoretical performance are also shown in the same figure with solid lines. Both cases are displayed for 1PE and 4PE. The segmented line displays the actual simulation in a Sun Sparc-2 station, which are comparable to one of a single PE. The speedup of the theoretical iteration (solid line) and actual runs (dotted line) are displayed in figure 17.

The speedup equation (EQ 16) indicates that the communication time is $O(n)$ while the computation time is $O(n^2)$. Thus the communication overhead is at the minimum when n is large. The percentage of communication overhead as actually measured (dotted line) and theoretically measured (solid line) are displayed in figure 18.

FIGURE 17

Speedup time ratios for a single iteration in the matrix solver on a C30 multiprocessor (dotted-line) and the theoretical performance (solid-line)

FIGURE 18

Communication overhead for the iteration matrix solver using 4 PEs (dotted-line) and the theoretical performance (solid-line)

We can conclude that although the computational size granularity can be made quite small with 4 processors (i.e. with $N = 4$ and $n = 4$ computational task = 6 us) and (the communication task overhead = 800 ns), the metric computation/communication ratio = $6/0.8 = 7.5$ is still considered reasonable to obtain a relative speedup of 3.5. This architecture is used as the testbed of our distributed WR circuit simulation algorithms. In this slotted-ring architecture, the way processors communicate is exclusively through message-passing, since PE only has access to its own local memory. Therefore, this parallel hardware is included in the description of distributed memory systems.

3.3 Parallel Processing Circuit Simulation using Direct Methods

As mentioned in Chapter II, circuit simulation at the electrical level is a very time consuming process. The profiles of the SPICE circuit simulator demonstrates that MOS model evaluation and the matrix solution are the most consuming part of the simulation runtime.

Much attention has been given to accelerating these two steps in the Newton-Raphson iteration step. Therefore, techniques are presented here that exploit the parallelism inherent to these two steps.

There are two approaches to accelerate the circuit simulation using DM: one is to parallelize the SPICE code, and the other uses parallel processing techniques by applying circuit decomposition using tearing methods.

Before trying either approach it is important to introduce Amdahl's law because it provides the theoretical upper boundary when some codes of the program are strictly sequential.

If N is the number of processors, s is the amount of time spent (by a serial processor) on serial parts of the program, and p is the amount of time spent (by a serial processor) on parts of the program that can be done in parallel (see figure 19), then Amdahl's law says that speedup is given by

$$speedup = \frac{(s + p)}{\left(s + \frac{p}{N}\right)} \quad (\text{EQ 17})$$

If we normalize the total time $s + p = 1$, then

$$speedup = \frac{1}{\left(s + \frac{p}{N}\right)} \quad (\text{EQ 18})$$

This equation can also be written as:

$$speedup = \frac{N}{Ns + 1 - s} \quad (\text{EQ 19})$$

Many sections of the program code have to be executed sequentially. If there are N processors executing a parallel loop, then a speedup of N is possible as long as $Ns < 1$. Therefore to accomplish this, the sections of serial code of a program must be as short as possible. For example, if 14% of the code is executed sequentially, only a 4x speed

FIGURE 19

Amdahl's law graphical representation

advantage can be obtained for an 8 processor system. For $N \rightarrow \infty$, the speedup is limited to the reciprocal of the fraction of the computation that must be done sequentially.

Parallelizing the code of a program is usually a real challenge. The implementation must be carefully tailored to keep most of the processors busy throughout the simulation, while minimizing the overhead from the parallelization.

Three different types of code are distinguished in parallelizing the code of a program:

- 1) Serial code: These are portions of the code which must be executed on a single processor while no useful work is being done on the other processors.
- 2) Critical or Protected code: These are portions of the algorithms which can only be executed by a single processor at any given time, though other processors may execute different tasks in parallel within the protected code.
- 3) Parallel code: These are portions of the algorithm which can be executed independently in parallel with any task.

The main objective i.e., the conversion of any algorithm for parallel execution is the minimization of serial and critical codes.

Another consideration in optimizing a code for parallel programming is the granularity problem. The algorithm must be able to partition computational tasks evenly among the available processors so that all processors are kept busy all the time. If the loading is uneven between processors, the mismatch between their executions requires more time to execute the code for its data. This results in the other processors standing idle waiting for the slowest process.

3.3.1 Parallel Model Evaluation

At each Newton-Raphson iteration, the model formulation module is called to perform a model evaluation consisting of two steps: (1) the evaluation and (2) the stamping of the various devices in the circuit.

The evaluation phase involves the linearization of each non-linear device in the circuit. Once the node voltages of a particular device are known, evaluation of the linear companion network for the device can be performed independently of all other devices in the circuit and is therefore ideally suited for parallel processing.

The second step, matrix stamping, corresponds to the accumulation of the Jacobian and the RHS vector. Because this step requires writing to memory locations which may be shared by several elements, synchronization is required so that only one processor can write into the matrix at a given time. This can lead to memory contention in a shared-memory architecture and can decrease the efficiency of the parallel system.

The stamping process can be broadly divided in two groups: lock-based and barrier-based methods. In the first group, synchronization of the matrix (per element, per row, or one for the entire matrix), the device equations can be overlapped with the stamping process. The second group transforms the distributed lock synchronization points into one or more barriers which separate the sequence of evaluation/stamping operations, that is, the evaluation and stamping steps do not overlap. Different methods of synchronization are implemented in [8,77,51,2,23].

3.3.2 Standard Parallel Sparse Matrix Solution

In direct method circuit simulation, the solution of the Jacobian matrix is usually performed using LU factorization followed by forward elimination and backward substitution.

Circuit matrices are usually very sparse (about 4 nonzero entries per row or column), diagonally dominant, non-symmetric, and not positive definite. Iterative methods can exhibit a high degree of parallelism, but there are many cases for which there are severe convergence difficulties.

The sparse matrix solution with LU decomposition is difficult to parallelize, primarily because of the irregularity of the structure of the matrix. The LU decomposition algorithm has a sequential dependency and the amount of concurrent work which can be done at each step in a sparse matrix is small.

In SPICE, the LU decomposition can be performed with pivoting. The goal of a pivot-ordering algorithm is to maintain sparsity by reducing the number of fill-ins, therefore minimizing the total number of matrix operations. In a parallel processor the goal of the ordering is to reduce the total time required to perform the decomposition by increasing the number of pivots that can be computed in parallel [76]. If attempts are made to minimize the completion time by maximizing parallelism, more fill-ins are generated and the number of operations and possibly the completion time would increase. An heuristic algorithm will find the shortest depth in the triangulation matrix assuming that the number of processors is unlimited.

Experimental results using a commercial shared-memory multiprocessor, the Alliant FX/8, shows a maximum speedup of 4.5 using 8 processors [77] for a 330 MOS-FET transistor benchmark circuit. [77] also proves that its performance is limited by the

degree of parallelism in the sparse matrix solution. For this benchmark circuit it is shown that the model evaluation has 5 times more parallelism than the matrix solution if there are an unlimited number of processors.

3.3.3 Parallel Circuit Simulation using Tearing Methods

A method of solving large systems of equations, called tearing, has been investigated in a number of applications. Tearing does not necessarily save computations as compared to direct sparse matrix methods, but the subnetworks are handled independently and parallel processing can also be exploited. In the case of identical subnetworks, tearing may lead to savings in storage as well as in computation time. It also becomes important in digital circuit simulation where only a small proportion of the subnetwork changes.

In the node tearing method, the network is decomposed into smaller subnetworks by selecting several nodes as “Local Datum Nodes” (LDN) [78]. By gathering the rows and columns corresponding to LDNs around the border of the matrix, the circuit matrix becomes a BBD structure as shown in Figure 3.14a, where M_i expresses the circuit submatrix of the subnetwork N_i , P_i and Q_i^T refers to the interconnections between the nodes in the subnetwork N_i and the LDNs, and Z is the interconnection matrix of LDNs.

As a result, several processes of the matrix solution can be performed in parallel at the subnetwork level except for the interconnect matrix Z . Furthermore, node tearing may decrease the number of fill-ins since they do not propagate themselves out of each block submatrix. Partitioning the circuit into subcircuits or blocks greatly reduces possible memory update conflicts. Possible conflicts only occur when a subcircuit load enters in to the interconnect matrix. The only serial tasks are the loading of each subcircuit contribution to the interconnect matrix and the solution of the interconnect matrix. Therefore, the goal is to minimize the size of the interconnect matrix Z .

FIGURE 20

(a) BBD structure (b) nested RBBB structure

When node tearing is performed recursively, the circuit matrix is transformed into the RBBB structure as shown in figure 20b. This method has shown improved performance compared to the BBD structures. Since the submatrix Z corresponding to the LDNs is dispersed, more effective matrix solutions may be performed. One way to achieve the RBBB structure is to use nested BBD partitioning. The objective of this approach is to minimize the sizes of the interconnection set and the largest subcircuit at each level. The dependency relationship of factorizing a nested BBD matrix has the structure of a tree. The root of the tree is the set of the interconnection nodes of the first level partitioning. The leaves of the tree are the internal nodes of the non-partitioned subcircuits. The descendents of the tree must be factorized before the ancestors to ensure the correctness of the parallel factorization process. When the coefficient matrix has a nested BBD structure as depicted in figure 20b, each submatrix will cover all the border blocks corresponding to the diagonal submatrix. The precedence relationship among these submatrices can be represented by a task-dependency graph which is illustrated in figure 21.

FIGURE 21

Task graph representation of the RBB structure of figure 20

This decomposition method has been used successfully in [41] using hierarchical decomposition at the digital gate level, where the output node of each gate is selected as the LDN.

A similar tearing approach is used by [9] in which the circuit is divided into many small subcircuits that are dynamically allocated to processors when they become free. This work involves two different architectures: a Sequent Balance 8000 and the Alliant FX/8. Both systems have a common memory shared by all processors. The most successful decomposition uses heterogeneous partitioning by small functional units which correspond to the natural hierarchy used by the circuit designer and do not require circuit flattening. Cox found better than 90% parallelism with circuits when more than 4000 nodes were used, but these tests were run on a limited number of processors.

Similar work performed in a Sequent Balance 21000 parallel computer with 10 processors [5] shows that the performance start to roll off at 8 processors. The maximum speed attainable with 10 PE was 6.8 in a circuit with 470 nodes. This hierarchical partitioning was also implemented in a distributed parallel processor, the iPSC system with 16

processor elements, showing a maximum efficiency of 75% with 8 processors. The communication overhead increases when more processors are added, reducing the efficiency of the system.

3.3.4 Parallelism issues for Direct Methods

Parallelism exploitation using exclusively direct methods is still a big issue in the CAD community. Although the parallelization of the matrix load is relatively straightforward, the linear equation solution phase is still known to be a difficult problem to parallelize due to pivot dependency and the substitution process. Nevertheless, these techniques are being used in combination with other relaxation algorithms when processors are idle waiting for other processors to complete their work [45] [20]. Since these tasks are known to be fine grain, special dedicated hardware is required to handle the computation/communication trade-off.

3.4 Parallel Processing in Relaxation Methods

A higher degree of parallelism can be achieved using relaxation methods, mainly because the ODE equations in WR, or non-linear equations in ITA, become decoupled. Parallelism is usually exploited at the block level where the matrix/rhs load, LU decomposition and linear equation solution for each block can potentially be performed concurrently. A good review of task parallelism using relaxation techniques is presented in [56]. The amount of parallelism available depends on the particular relaxation scheme used and the synchronization points that are introduced for architectural or programming reasons.

The first task in a relaxation method is to partition the network in small tightly-coupled subcircuits as shown in figure 22a. The relationship between the subcircuits is defined by a partial ordering which is generated by extracting coupling information from

the subcircuit graph produced by the partitioner, as shown in figure 22b. The subcircuit graph contains a vertex for each subcircuit and the edges represent dependency relationships between the subcircuits. There is a directed edge from vertex i to j if subcircuit j contains an equation that depends on the value of a node voltage in subcircuit i . This is the usual representation of MOS circuits, in which the signals are bidirectionals when the internal capacitances in the transistor model are incorporated. These signals are usually more "dominant" from gate to source and gate to drain, though not so in the opposite direction.

The partial ordering is generated by considering the dominant edges in the graph and produces a directed acyclic graph as shown in figure 22c. This graph defines the relaxation scheme by specifying the data dependencies to be enforced within one iteration. This is the typical data dependence of a single Gauss-Seidel iteration. As can be seen from figure 22c, subcircuits s_1, s_2, s_3 belong to the first rank level ($m=1$) based on the distance in terms of the directed edges from an input source. Subcircuits s_4, s_5 belong to the next rank ($m=2$) and subcircuit s_6 correspond to the last rank level ($m=3$). Therefore, this Gauss-Seidel graph has three ranks, and any subcircuit which belongs to the same rank can be processed simultaneously. If we assign each subcircuit s_i in the same rank to a different processor, the maximum number of processors active at the same time is equal to 3.

The Gauss-Jacobi method is generated by placing all the subcircuits in a single rank and ignoring the directionality of the circuit within a single iteration. This is illustrated in figure 22d in which all subcircuits $s_1 \rightarrow s_6$ are on the same rank ($m=1$). Therefore, a maximum of 6 processors are active at the same time using the G-J method.

Although the G-J produces more parallelism than the G-S, the G-S method is the algorithm doing the most useful work at any given time.

FIGURE 22

Examples of Gauss-Seidel and Gauss-Jacobi partial ordering for one iteration. (a) original circuit partitioned (b) subcircuit dependency graph (c) Gauss-Seidel ordering $m=3$ (d) Gauss-Jacobi ordering $m=1$

The G-S method applied to parallel processing introduces artificial synchronization points at the end of each processing rank. Using such barrier synchronization can be a problem, because a large variation in subcircuit sizes may create load unbalance between processors and reduce the efficiency of the system. This is because large subcircuits require more CPU time to process than smaller subcircuits. One way to deal with this problem is to break up larger subcircuits into smaller ones so that the loading is more evenly distributed between processors. Since this operation may degrade the convergence speed of the whole simulation, careful circuit partitioning must be done. Another approach to circumvent this problem is to use a combined parallel model evaluation. When one processor becomes idle within a single iteration loop, this idle processor will help other processors to evaluate the elements of the subcircuit they are simulating. This dynamic combination will keep most of the processors busy, increasing the efficiency of the system [21].

3.4.1 Parallelism in Iterative Timing Analysis

The first program to try to parallelize ITA algorithms was MSPLICE [10][11], implemented on the Butterfly multiprocessor with 10 processors using a weakly chaotic relaxation. This program showed excellent efficiency of 70% with up to 10 processors.

MSPLICE uses the data partitioning approach to multiprocessing, where each processor performs identical functions on different parts of the circuit, rather than functional partitioning, where different functions would be assigned to different processors. A task in MSPLICE is defined as the evaluation of a single variable (in this case, a node voltage) for a single iteration. Therefore, the granularity of each task is approximately the same, which is desirable from the standpoint of load balancing. Since there are usually many more nodes in the circuit than there are processors in the system, more than one node may be assigned to each processor. A distributed scheduling algorithm is used for task assignment

whereby each processor is responsible for scheduling not only its own tasks, but certain tasks for other processors as well.

MSPLICE uses a weakly chaotic relaxation scheme to avoid synchronization during the iterative process. In this scheme, the new values of a given component are computed using whatever values are available for the external variables. That is, the computation is not required to follow any particular order, nor is a given component required to use a particular set of previous iterates in its evaluation. In fact, the external values could be recently computed values from earlier iterations. The advantages of this approach are that it removes synchronization from the iterative process and distributes the communication load more evenly over time. A global convergence counter, *GlobalRemainingNets*, is used to coordinate the processors at each time point. This variable is incremented whenever a node is scheduled for an iteration and decremented when the node is processed. When *GlobalRemainingNets* reaches zero, it indicates that all nodes have converged at the present time point. Therefore, the processors may proceed to the next time point. In MSPLICE, nodes can be assigned to processors using either static or dynamic allocation. A number of trade-offs exist in each approach [11][23]. For example, in a static allocation scheme, the architecture of the Butterfly suggests that adjacent nodes in a circuit be assigned to the same processor to keep the number of remote memory references to a minimum. However, latency exploitation usually reduces the number of tasks available during the iterative process. Since adjacent nodes in the circuit tend to be latent at the same time, some processors may run out of tasks while others still have many tasks to process. This creates an imbalance in the processor loads and suggest that adjacent circuit nodes should be placed on different processors to reduce the likelihood of all the nodes on one processor being latent.

A dynamic assignment strategy is preferable to a static assignment so that load balancing can be performed more effectively during execution. However, a significant overhead may exist in moving circuit data from one processor to another, unless a copy of the entire circuit exists on each processor, one which is memory inefficient for large circuits. Another issue in dynamic assignment is that of performing dynamic load balancing. In a more recent version of MSPLICE [24], each processor schedules new events on the processor with the fewest events in its queue. This involves searching a global data structure to determine the processor that has the shortest task queue. As the number of processors increases, contention associated with the global data structure could degrade the performance of the program. In fact, since N entries in the global data structure are searched by a possible N processors, the overall contention grows as proportionally to N^2 in this implementation. However, since a task requires a significant amount of time to complete, the queue contention does not pose a major problem with up to 100 processors [24]. In the current implementation, the dynamic scheduling algorithm shows the best performance, with an approximate efficiency of 80% using 32 processors, although the benchmark circuit presented in [24] has very little feedback. More detailed analysis of relative speedup would therefore be required to indicate the effectiveness of the multiprocessor-based circuit simulator.

Saleh et al [54] exploit another parallel implementation of the ITA algorithm using chaotic relaxation methods, on a shared-bus, shared-memory architecture where a limited number of processors are available, called PSPLICE. This program was implemented on the Sequent Balance machine which uses basically the same algorithms as SPLICE 3.1. A number of experiments were performed using the PSPLICE program to compare G-J, G-S, and chaotic relaxation methods.

The first step was to try a Gauss-Jacobi-like algorithm by processing as many subcircuits in parallel as possible. This is referred to as the “First-Available-Task” approach. In this approach, the subcircuits are obtained from the head of the subcircuit queue and processed immediately using whatever values are available for external variables. While the method is not pure Gauss-Jacobi (since there may be updated information available in some cases) the approach becomes closer to Gauss-Jacobi as more and more processors are added. In fact, the method is weakly chaotic with the guarantee that at least one input to each subcircuit in the task queue will have a recent iteration value. The results of this implementation indicate reasonable efficiency can be obtained using this approach. The speedup factors on 8 processors are between 4 and 5, which translate to the efficiency of approximately 50-60%.

The next step was to try to improve the performance using a Gauss-Seidel algorithm. Although the Gauss-Seidel algorithm is sequential in nature, there are a number of opportunities for parallelism. Consider, for example, the circuit graph shown in figure 22c. The width of the graph is the maximum size of any independent subset of tasks and it is this aspect that provides the parallelism in the Gauss-Seidel algorithm. The amount of parallelism available using this approach depends on the topology of the circuit. That is, this approach will work quite well on circuits having rather wide graphs but will not be as effective on circuits with narrow graphs. Another factor which limits the amount of parallelism is latency exploitation. Some subcircuits may be latent while others may converge quickly during the iterative process so that further processing is unnecessary. However, if the circuit is large enough, the Gauss-Seidel method may be still be efficient. As in the First-Available-Task algorithm, each subcircuit task can also be subdivided into a number of model evaluation subtasks to generate more parallelism. If each subcircuit is large, there may be enough work to keep all processors busy even if the subcircuits are processed one-at-a-time. Therefore, for a machine with a small number of processors, it may

be possible to maintain the strict Gauss-Seidel ordering and obtain a speed-up relative to the uniprocessor version.

When the Gauss-Seidel version was implemented in PSPLICE, the results were not very promising, due to the fact that dynamic ordering was not being used and continuous checking of the subcircuit queue was undertaken. For a particular circuit with a not very wide graph and many feedback paths, the multiprocessor system is not being fully utilized when it executes in Gauss-Seidel mode, where the number of busy processors rarely reaches eight.

Two alternatives exist to improve the performance: the granularity of the computation can be increased, or the time that processors are blocked can be reduced. One way to generate more work is to first try to find a task which is ready to fire, and if none is found, to simply take any task from the queue. This could be from the head of the queue or any other part of the queue. The run times for this modification are comparable to the First-Available-Task version.

Based on the results obtained in PSPLICE, the First-Available-Task version has been shown to be the easiest to implement and appears to be the most efficient [54].

Another important application of nonlinear relaxation techniques has been reported on the Connection Machine [68]. This processor belongs to the family of massively parallel processors which have recently become feasible because of the decreasing cost of VLSI chips. They provide some interesting alternatives to conventional parallel processing methods. The connection machine consists of up to 65,536 processors, interconnected on both a hypercube and a two dimensional grid. The hypercube provides communications for algorithms with an irregular message structure, while the grid is much more efficient for algorithms that map easily onto a corresponding two dimensional grid.

Each processor has 4096 bits of memory and one bit data path. The control is Single Instruction Multiple Data (SIMD), which means that all processors must either be executing the same instructions or nothing at all.

A Gauss-Jacobi relaxation at the nonlinear equation level was implemented in this machine with 32,768 processors, using node based decomposition and a single step of a Newton-Raphson iteration combined with the relaxation iteration. In such an experiment, the numerical integration is done using the trapezoidal rule. This maps with a much difficulty onto the Connection Machine architecture, but has the disadvantage of possible slow convergence for tightly coupled circuits. The data for the algorithm is broken up so that all the circuit nodes and devices are allocated to a unique processor, so that the algorithm can proceed as follows. First, for each iteration step, send (using message passing) all the node voltages from the processors containing the circuit nodes to the processors containing the devices. Second, evaluate all the devices by selecting all the processors containing a given device model and executing the evaluation routine for that model. Third, each of the devices sends the results of their computation back to the processors containing the circuit nodes. And fourth, all the processors containing circuit nodes are again selected, the code for completing the iteration step is executed, and the node voltages are updated. This process is repeated until the relaxation iteration converges and then advances to the next time step.

The experimental results show that the execution time is independent of problem size for circuits without tight coupling. Because of this fact, this architecture is only useful for very large problems: it is extremely slow for small problems when compared to uniprocessors. But for large problems, the performance of the current implementation is approximately equal to that of conventional mainframe computers.

In order to improve the convergence properties for circuits with tight coupling, either direct methods or relaxation methods using partitioning must be used. Partitioning, however, breaks the problem up into irregular sized blocks, which do not map well onto the Connection Machine architecture because only matrices of the same size can be solved simultaneously. For direct methods, the main problem is again to solve a large sparse matrix. While this is still a difficult problem, there are some algorithms using nested dissection that look promising for this purpose [48].

3.4.2 Parallelism in Waveform Relaxation

From a circuit viewpoint, the WR algorithm partitions the circuit into small subcircuits. Each subcircuit is simulated over the entire given time interval with an independent time step. Relaxation in function space is performed to take into account the coupling between subcircuits. Thus, task granularity at the WR level is characterized as being of much larger grain than the ITA algorithms, assuming similar partitioning schemes are used. Therefore, distributed memory architectures are being used to exploit parallelism at this level [22][71][26], assuming that the metric computation/communication ratio is large enough to reduce the communication and synchronization overheads.

The simulation of a subnetwork in one WR iteration and in a given time interval could be executed on a single processor and several of these jobs can be done concurrently using multiprocessor systems. In the simplest algorithm, a static list of subnetworks is made based on the signal flow through the circuit [73]. All subnetworks can then be simulated independently on different processors. Each processor takes the next subnetwork from the list, without concern as to whether the inputs of that subnetwork are computed or not. This is in fact a mixed Gauss-Seidel Gauss-Jacobi algorithm. The problem with this algorithm is that it needs more iterations to reach convergence than a Gauss-Seidel algorithm. Its advantage is that there is a great deal of parallelism.

Odent's implementation uses a pure Gauss-Seidel algorithm, showing that a subnetwork can be simulated if all its inputs are computed in the current WR-iteration and all its fanout nodes are at least from the previous WR-iteration. In this case, when a subnetwork becomes available for simulation, it is placed on the stack. When a processor becomes free, it takes a subnetwork from the stack and simulates it. The advantage of this method is faster convergence, but again the drawback is less algorithmic parallelism. The use of a global stack with available subnetworks for simulation, and of a data flow principle provide an efficient way to overcome the problem of load balancing. Jobs are not statically assigned to a processor but are dynamically partitioned over the processors when they become free during the execution of the program.

Newton et al [41] introduced two new methods to increase algorithmic parallelism, called dynamic decoupling of the circuit and pipelining of time segments.

The first method is based on the dynamic decomposition method in the program SWAN [12], based on circuits with a two clock phase cycle, to control some pass transistors as shown in figure 23.

FIGURE 23

Dynamic decoupling using two phase clock cycles in a chain of inverters

When the clock PH1 is low, node C is decoupled from node B. This means that changes in the state of node B will not affect the state of node C. When PH1 is low, C can be considered as an external input in the scheduling algorithm. This means that the

subnetwork with node C as input can be simulated before C is simulated. Since node G is also a decoupled node, the number of subnetworks available for simulation at the same time increases from one to three. When PH2 is low, the same considerations apply to nodes E and I, and the number of available subnetworks for simulation is also three instead of one. With this algorithm, the simulation interval is partitioned into time segments. In each time segment the decoupled nodes are searched, the simulatable subnetworks are placed on the stack and the simulation is started up.

If there are feedback loops in the circuit, the WR process may take many iterations to converge. The problem was solved by [45] by searching for the strongly connected components (SCC) in the subnetwork graph. Two subnetworks are placed in the same SCC if there exists a direct path from subnetwork A to subnetwork B and from subnetwork B to subnetwork A. The algorithm used to find the SCC's is a depth-first search algorithm [66] of linear complexity.

The simulation of the circuit is divided in two parts:

- The GroundWave (GW) phase.
- The Waveform Relaxation (WR) phase.

In the GW-phase, the SCC's are simulated by local relaxation. When a simulatable SCC is found, the subnetworks of the SCC are simulated iteratively until convergence of the waveforms has been reached. To simulate the subnetworks in the correct order, the feedback loops are opened and the subnetworks are ordered in a preprocessing phase. In the WR-phase the simulation of all subnetworks is iterated until convergence of the circuit has been reached.

Since for finite state machines, nearly the whole circuit can be considered as one large SCC, different processors must be able to simulate different subnetworks of the same SCC in parallel to be efficient.

It is also possible to start the WR phase before the end of the GW phase. When the ground waveform of two levels of SCC's have been computed, the WR iteration can be started. This pipelining of the GW-phase and the WR-phase, called "the overlapped WR iterations method", was also included in the dataflow approach.

For small circuits the speedup can be quite poor, due to the limited number of subnetworks that are simulatable at the same time. To speedup the simulation it is possible to execute the subcircuit simulator, based on the direct method, in parallel.

Since in a relaxation method the circuit is partitioned into smaller subcircuits, the major portion of the computation in the subcircuit simulator consists of the element evaluation. For each timestep all elements can be computed in parallel. Since the stamps of the MNA matrix can be in the same entries for several elements, some synchronization is needed. The speedup will be limited since only one part of the direct method is done in parallel. However, in combination with the parallelism at the subnetwork level, it is a good approach, exploiting parallelism at three different levels: SCC's, subnetworks and elements.

The parallel element evaluation implemented in [45] is based on a shared stack. On that stack are placed all elements that have to be evaluated for the current timestep of a subnetwork under simulation for a given processor. If another processor becomes idle, i.e. if there is no simulatable SCC or subnetwork for that processor, it can help another processor with the element evaluation. The idle processor can then look at the stack of available elements for evaluation by other processors. If it finds such an element, it is evaluated and

its stamp is added in the MNA (Modified Nodal Analysis) matrix. It is obvious that the function **take_element_from_stack** and **place_stamp_in** are protected with lock operations. Figure 24 shows the scheduler for parallel element evaluation. A variable **new_subn** can stop the parallel element evaluation if there are new simulatable subnetworks after the simulation of a subnetwork on another processor.

Note that there is no static partitioning of tasks to processors. Each processor can activate SCC's, simulate subnetworks and evaluate elements.

Another method to increase parallelism uses pipelining of computed waveforms. Although [75] uses such a method in timepoint pipelining (TPP), the simulation time on one processor is 50% longer due to overhead. To reduce the overhead, time segment pipelining is used (TSPL). This new method will be explained using the chain of inverters

```

Scheduler () {
  While (circuit not converged) {
    While (scc) ... {
      While (subnetwork) ...{
        While ( elem = take_elem_from_stack () ) {
          evaluate (elem);
          place_stamp_in (MNA_matrix);
          if (new_subn) break;
        }
      }
    }
  }
}

```

FIGURE 24

The three level scheduler

shown in figure 23. In the Gauss-Seidel relaxation method and taking into account the directionality of the circuit, subnetwork 2 can only be simulated if subnetwork 1 has been simulated. This is because the internal node of subnetwork 1 is an input of subnetwork 2. This condition is in fact too restrictive. When the first subnetwork has been simulated over a certain time segment, the simulation of the second subnetwork can also be started within that time segment. At the same time, the first subnetwork can be simulated in a new time segment.

In the time segment pipelining method the whole simulation interval is divided into a number of time segments. The simulation of subnetworks is done over only one time segment. A subnetwork is now simulatable if all inputs are computed in the same time segment for this waveform iteration and if all fanout nodes are computed in the same time segment as for the previous waveform iteration. Feedback inputs of the subnetwork are considered as fanout nodes.

These algorithms were implemented on the Sequent Balance 8000, with the simulator CSWAN. The results show that for 8 processors, the simulation time is dependent on the size of the time segment. However, the size is not very critical except for very small time segments, where the scheduling and communication cost are becoming so large that the increase in parallelism is not large enough to improve the speedup.

The biggest advantage is shown by the use of the time segment method on a given number of processors, where the speedup factors can be high, even for small circuits, and the overhead is limited.

3.5 Waveform Relaxation in a Distributed Simulation Environment

As shown in the previous section, most of the research effort was focused in parallel processing to improve the performance in circuit simulation. The architectures that have been used to accelerate these techniques are in general based on shared memory architectures. Today, with the advent of high performance workstations and high speed interconnection networks, distributed processing offers a cost effective and attractive solution to the circuit simulation problem. Workstations available in most universities and research centers can be effectively used for distributed simulation, resulting in shorter simulation times without any additional investment. As well, workstations are general purpose, easy to use computers and hence can be used for a variety of applications as compared to dedicated parallel machines. Still, the major drawback of using distributed machines for circuit simulation is the high overhead for interprocessor communication.

Earlier work by Wen [71] shows that WR ported in Sun Sparc workstations can generate some parallelism if careful partitioning and allocation algorithms are performed. The work is based on the conservative WR synchronization techniques and the number of workstations was limited to only four. Speeds up to 2.4 times were obtained for large circuits (3000 transistors and up) and very loosely-coupled subcircuits. For medium size circuits (100 to 400 transistors) no speedup was achieved, because the communication overhead overshadows the functional parallelism.

Current research work concentrates on the WR algorithms in a distributed environment. The main reason why a distributed simulation algorithm is pursued is that a centralized algorithm in a shared-memory paradigm is less scalable, where the global synchronizer which orders event simulations is often the bottleneck. With a distributed algorithm, synchronization is performed by individual processors. Thus, the WR algorithm implementation is based on asynchronous subcircuit simulation, which

communicates solely via message-passing. The development of the WR algorithm depends on several issues to optimize the parallelism in these selected distributed architectures, which will be addressed in the next chapters.

We believe that careful partitioning and allocation techniques can create enough task granularity such that the ratio of the metric computation and communication time is maintained high during the complete simulation interval. Other issues, such as distributed scheduling algorithms, are required to maintain the Gauss-Seidel ordering as much as possible, such that the functional parallelism is exploited to maximum. Nevertheless, in many cases spatial domain parallelism is not sufficient, specially for circuits that have a very narrow task dependency graph. Temporal domain parallelism is necessary to pipeline the simulation in these cases, such as the TSPL techniques. Mixing both kinds of parallelism at the WR level seems a viable technique to optimize for parallelism in a distributed simulation environment, which is considered the goal on this research. The WR distributed algorithms have been ported to both available architectures: the slotted-ring and the Sun Sparc-2 workstations.

The slotted-ring architecture will be used as a vehicle to map our algorithms as dedicated hardware, which may provide optimum speedup, since there is no operating system and the communication latency can be at minimum. The limitation of this system is that currently only four processors are available for simulation, therefore the scalability of the algorithm can not be fully tested. The other constraint is the size of main memory, currently 512 Kwords. Only small to medium size circuits with 200 transistors can be used for simulation.

The Sun workstations are more general purpose, multicomputer networks, which communicate on an Ethernet bus at 10 Mbits/sec. The message passing communication protocol is based on the Unix TCP/IP. This low bandwidth communication is the penalty

paid for generality, since these workstations are aimed at general purpose computations. In the Department of Electronics at Carleton University a variety of Sun workstations are available for research use. These include Sparc-2, Sparc-10 and Sparc-5 workstations that can be used for our distributed framework in circuit simulation. These multicomputer networks are preferable for the algorithm scalability test, and also offer a large memory space available.

3.6 Conclusions

In this chapter, different parallel architectures and algorithms were presented in the context of circuit simulation. Direct and relaxation methods were also addressed, identifying the parallelism of these algorithms. A literature review of different techniques used in the parallel electrical simulation level, in which different approaches are taken, was presented. In attempting to parallelize the standard circuit simulation methods, a bottleneck is still found in the irregular structure of the sparse matrix. The two best approaches were those that reordered the matrix in BBD or RBBD structures. Other techniques, such as iterative methods, can exhibit a high degree of parallelism, as relaxation methods do. The Gauss-Seidel relaxation is difficult to parallelize because of the ordering constraints; therefore not many processors can be kept busy, especially when the subcircuit dependency graph is not very wide. Gauss-Jacobi relaxation can also be used and provides a large degree of parallelism because the constraints are removed. Its disadvantage is that it takes many more iterations to converge than the Gauss-Seidel. A trade-off should be applied based on the architecture being used and the amount of parallelism available in the algorithm.

In this chapter, a parallel, iterative algorithm for solving large, dense linear systems of equations on a ring architecture is presented. Performance is evaluated

analytically and compared with the actual simulation on the parallel machine, with the results showing that the architecture can effectively implement the WR distributed algorithms.

Although parallel algorithms using the shared-memory paradigm are effectively being used to accelerate the relaxation methods, the applications start to saturate with the number of processors, mainly because of the central scheduler bottleneck and memory contention. On the other hand, although their algorithms are more difficult to implement, in a distributed-memory paradigm (global variables are not existence in this paradigm) the potential concurrency is much more natural, since objects execute their events from their own queue. Therefore, this collection of queues taken together constitutes a distributed event queue. Causality must strictly be preserved through the simulation through strict adherence to the rule that events are processed in non decreasing timestamp order. The framework of this research is laid out based on the distributed WR circuit simulation, which will be expanded in more detail in the next chapters.

IV Spatial and Temporal Partitioning in a WR Distributed Simulation Environment

In this chapter, we discuss the simulation model of the Waveform Relaxation method in the distributed simulation environment. Several issues must be considered to obtain a good circuit simulator performance in a distributed architecture. These are: circuit partitioning, subcircuit allocation, ordering, and scheduling; as well the granularity problem. Since our goal is to minimize the computational expense, these issues must be tuned in such a way to optimize parallelism, communication and synchronization overhead. Trade-offs normally exist that take all these issues under consideration. In this chapter we mainly concentrate on grain size determination, and circuit partitioning for a parallel distributed environment.

4.1 Spatial and Temporal Partitioning in the Waveform Relaxation

Waveform relaxation techniques need to decouple the equations in both the spatial and temporal domain to accelerate circuit simulation. Spatial domain decomposition is needed to decouple the circuit into subcircuits so the interaction between them is minimized. Effective time window sizes for WR (temporal domain decomposition) is another crucial issue for the fast convergence of the WR methods. A good combination of both partitioning techniques may provide effective solutions in the simulation of circuits.

4.1.1 Spatial Domain Decomposition

All relaxation schemes have spatial partitioning in common, in which the objective is to decompose the system of equations in a number of loosely coupled components so that their variables can converge with the minimum number of iterations. Therefore the convergence speed on WR depends on the degree of coupling between the equations in the system and the order in which the equations are solved. This last issue is related to the scheduling technique offered by the simulator, which will be discussed in Chapter V.

Node by node decomposition, initially proposed in WR, has proved to be very inefficient when the circuit contain tightly coupled-nodes [74]. Thus, a good partitioning scheme is required that identifies strongly-coupled components in the circuit automatically and groups them together to form subcircuits. Since the variables associated within the subcircuit are assumed to be tightly-coupled, the subcircuits can be solved using Direct Methods, and the WR method is applied between subcircuits.

FIGURE 25

Illustration of spatial partitioning

Figure 25 illustrates spatial partitioning and the interaction between subcircuits. These subcircuits in the WR simulation require constant exchanges of node voltage information between the coupling at the boundaries. If subcircuits are located in different processors, the information is exchanged through message-passing. Only those waveforms of nodes that contribute to the inter-subcircuit interactions have to be exchanged.

Many partitioning techniques have been used in the circuit simulation, but those that incorporate the coupling effect are the DC unilateral subcircuit partitioning and Norton Admittance partitioning, which will be described next.

4.1.1.1 DC Unilateral Subcircuit Partitioning

This partitioning is based on the topology of the circuit. The subcircuits are obtained by breaking the gate-source and gate-drain connection of MOS transistors. Nodes which are coupled over drain-source paths are merged together. Other two terminal elements such as resistors, capacitors, voltage sources, etc. can be included in the path of a strong connected subcircuit. When a unidirectional MOS model transistor is used (neglecting the gate-source and gate-drain capacitances) and no global feedback is assumed in the circuit, only one relaxation iteration is necessary to achieve the solution. For more realistic models, this method can increase the number of waveform iterations. This method is the most popular WR method for circuit partitioning because it creates a large amount of small subcircuits. However, it is not an ideal method because it explores the topology of the circuit but not its electrical characteristics such does the Norton admittance partitioning method.

4.1.1.2 Norton Admittance Partitioning

This method tries to estimate how fast the WR iterations will converge. White et al [73] shows that the speed of convergence is controlled by the contraction factor γ , in the following way:

$$\|x^{k+1} - x^k\| \leq \gamma \|x^k - x^{k-1}\| \quad (\text{EQ 20})$$

The goal is to compute a worst-case estimate of the resulting convergence factor γ for each pair of circuit nodes. The two circuit nodes are clustered if the estimate is higher than a user-specified threshold value α . The coupling of the circuit nodes is modelled as a nonlinear admittance Y_{12} connecting the circuit nodes as shown in the generic coupling model in figure 26.

FIGURE 26

Generic coupling between two nodes in a circuit

The generic coupling between two nodes in a circuit are represented by a conductance and a capacitance in parallel, as shown in more detail in figure 27. The values of g_1 and g_2 are the Norton equivalent conductances seen from each node looking back into the rest of the circuit. The capacitor C_{12} represents the total capacitance between node 1 and node 2, and capacitors C_1 and C_2 represent the total node to ground capacitance at each respective node.

FIGURE 27

Linear circuit considered for partitioning

Since MOS circuits are non-linear, each non-linear device must be replaced by a linear equivalent device. For a static partitioning strategy, worst-case conductance and capacitance values are used when replacing each non-linear device with a linear one.

Using the Gauss-Seidel relaxation equations for the circuit in figure 27 (using conductances only) results in the following equations:

$$v_1^{k+1}(g_1 + g_{12}) = v_2^k g_{12} \quad (\text{EQ 21})$$

$$v_2^{k+1}(g_2 + g_{12}) = v_1^{k+1} g_{12} \quad (\text{EQ 22})$$

The convergence factor, γ , is then defined as:

$$\gamma_{cond} = \frac{g_{12}}{g_1 + g_{12}} \cdot \frac{g_{12}}{g_2 + g_{12}} \quad (\text{EQ 23})$$

A similar analysis can be used for the capacitive coupling, where the convergence factor is:

$$\gamma_{cap} = \frac{C_{12}}{C_1 + C_{12}} \cdot \frac{C_{12}}{C_2 + C_{12}} \quad (\text{EQ 24})$$

```

 $g_{12} \leftarrow 0; g_1 \leftarrow 0; g_2 \leftarrow 0;$ 
foreach (conductive element between nodes 1 and 2) {
     $g_{12} \leftarrow g_{12} + \max(\text{conductance element overall});$ 
    remove the element from the circuit;
}

 $g_1 \leftarrow \text{sum}(\min(\text{Norton equivalent conductance of each element at node 1}));$ 
 $g_2 \leftarrow \text{sum}(\min(\text{Norton equivalent conductance of each element at node 2}));$ 
if (  $g_{12} / (g_2 + g_{12}) + g_{12} / (g_1 + g_{12}) > \alpha$  ) {
    place the two terminal nodes in the same subcircuit;
}

```

FIGURE 28

Algorithm for conductance partitioning

Although both conductance and capacitance clustering are done separately, a compound partitioning is the union of both partitions such that two nodes that belong to the same subnetwork in either partition has to belong to the same subnetwork in the compound partitioning. The union of both partitioning methods is based on the approximation used in the Jacobian matrix for a linear admittance with g and C in parallel, which generates a stamp of $(g + sC) \approx g + \frac{\alpha_0}{h} \cdot C$, where h is the integration step and α_0 is some coefficient of the integration formula. It can be seen that at DC (h large) the conductance part is dominant, but that at transients (h small) the capacitance will dominate the coupling.

To find the worst-case coupling, the internode admittance should be maximized and the node-to-ground admittance minimized. The following algorithm shown in figure 28 is used in both RELAX2.3 and iSPICE3 simulators for partitioning the circuit.

3.3.2	Standard Parallel Sparse Matrix Solution	51
3.3.3	Parallel Circuit Simulation using Tearing Methods	52
3.3.4	Parallelism issues for Direct Methods	55
3.4	Parallel Processing in Relaxation Methods	55
3.4.1	Parallelism in Iterative Timing Analysis	58
3.4.2	Parallelism in Waveform Relaxation	64
3.5	Waveform Relaxation in a Distributed Simulation Environment	70
3.6	Conclusions	72
IV	Spatial and Temporal Partitioning in a WR Distributed Simulation Environment	74
4.1	Spatial and Temporal Partitioning in the Waveform Relaxation	74
4.1.1	Spatial Domain Decomposition	75
4.1.1.1	DC Unilateral Subcircuit Partitioning	76
4.1.1.2	Norton Admittance Partitioning	77
4.1.1.3	Dynamic Partitioning	80
4.1.1.4	Hierarchical Partitioning	80
4.1.1.5	Load Balance - Convergence Trade-off Partitioning	82
4.1.2	Temporal Domain Decomposition	85
4.1.2.1	Fixed Time Window Decomposition	85
4.1.2.2	Dynamic Window Decomposition	91
4.2	Task Granularity in a WR Distributed Environment	91
4.3	A Portable Parallel Simulator	94
4.4	Summary	96
V	WR Parallelism in the Spatial and Temporal Domain, Scheduling and Task Allocation	98
5.1	Subcircuit Ordering	98
5.2	Task Parallelism and Scheduling in the Spatial Domain	100
5.2.1	Data-Flow Based Algorithm	101
5.3	Task Parallelism and Scheduling in the Temporal Domain.	105
5.4	Time-Segment Pipelining	108

2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

A similar algorithm is used for partitioning based on capacitances. A threshold parameter α is used to decide whether or not the nodes should be solved together or separately.

The exact Norton equivalent model seen by each node cannot be computed efficiently because it involves tracing paths from each node to all other nodes in the circuit. For efficiency, the depth of the conductance and capacitance computing process is truncated whenever the gate of an MOS transistor is encountered, since the conductance of an MOS transistor is zero in the worst case.

The problem with this algorithm is a pessimistic partitioning method that may result in unnecessarily large subcircuits. Therefore, the advantages of the relaxation are lost because most of the variables are solved with the Direct Method instead. There is also a disadvantage in a parallel system because large subcircuits create unbalanced loads between processors, reducing the efficiency of the system.

4.1.1.3 Dynamic Partitioning

This method is mainly used for bipolar circuits, since it is not possible to find a good static partitioning method for the circuit because the coupling between nodes varies by orders of magnitude. A dynamic partitioning method [37] uses a similar approach to Norton admittance partitioning, but it uses small-signal conductance and capacitance values, adjusting the subcircuits as these values change during the simulation.

4.1.1.4 Hierarchical Partitioning

The Norton admittance partitioning method implemented in SPLICE3 and RELAX2.3 is an efficient method of partitioning the circuit in subcircuits, because it decouples in such a way as to reduce the number of relaxation iterations between

subcircuits. However, when this method is used in a parallel machine, it is not efficient because the partition method creates large subcircuits. These are not suited to the parallel environment because they are not evenly distributed in all processors, creating highly unbalanced loads in the system, and lead to starvation of some processors.

CedarSim simulator use a two-layer partitioning approach, in which the original circuit is partitioned into a number of subsystems using the Norton admittance approach, by choosing a threshold parameter α_1 , in which α_1 is a low factor. The subcircuits created from the α_1 partitioning are called super-subcircuits which are assigned to different cluster processors in a hierarchical hardware (the Cedar multiprocessor system). Each of these super-subcircuits is further partitioned into smaller subcircuits by increasing the level of coupling α_2 , $\alpha_2 > \alpha_1$.

FIGURE 29

Hierarchical spatial partitioning with 2 layers

The creation of smaller subcircuits is allocated within the same cluster level. This partitioning method has the advantage of adjusting the degree of coupling based on architectural considerations. Weaker coupling is preferred between clusters because the number of relaxation iterations at this level is reduced, therefore considerably reducing the communication between them.

Tighter coupling is expected at the cluster level because more iterations are required between subcircuits, which is tolerated by the hardware requirements in the hierarchy. Figure 29 shows the hierarchical spatial partitioning with 2 layers.

4.1.1.5 Load Balance - Convergence Trade-off Partitioning

Although the DC unilateral and Norton admittance partitioning methods are well known methods to decompose the circuit in the spatial domain to accelerate the convergence simulation in uni-processors, they do not always perform well in parallel machines. Highly different subcircuit sizes may result by using the above schemes. Table 2 displays a typical benchmark circuit *Scdac* with its respective subcircuit sizes and simulation times using different α values for a single processor. Note that by using $\alpha = 0.1$, the partitioner creates only 9 subcircuits, therefore limiting the multirate properties of the WR algorithm, increasing the simulation time. Choosing a much bigger α value ($\alpha \geq 0.8$), the partitioner creates more subcircuits (60), at the expense of longer convergence times.

TABLE 2

Scdac runtimes and circuit partitioning using different α threshold values for a single Sparc-2 workstation

α	# Subckts	Avg # Nodes	Max # Nodes	Simulation Time (sec)
0.1	9	16.7	138	77.8
0.2	22	6.81	54	50.2
0.3	29	5.17	54	46.4
0.4	34	4.41	17	45.5
0.5	39	3.84	16	45.1

TABLE 2

Scdac runtimes and circuit partitioning using different α threshold values for a single Sparc-2 workstation

α	# Subckts	Avg # Nodes	Max # Nodes	Simulation Time (sec)
0.6	52	2.88	15	46.2
0.7	56	2.67	13	51.8
0.8	60	2.50	7	68.2
0.9	60	2.50	7	68.2

Also included in the table are the average and maximum number of nodes in a sub-circuit. It is also important to notice that in all benchmark circuits the trend is that the simulation time increases with the value of α . Choosing an $\alpha = 0.5$ provides a reasonable average number of nodes per subcircuit, but there are still a few subcircuits with a large number of nodes, which can be partitioned further. An iterative approach explored in [71] is used by reducing the size of these large subcircuits. The partitioning method selects those large subcircuits for further partitioning by repetitive increasing the α threshold value. The process terminates when an upper bound threshold is reached. For an upper bound threshold of $\alpha = 0.8$, it is possible to reduce the number of **Max # Nodes** in a subcircuit, i.e., for the *Scdac*, this value was reduced from 16 to 7. Table 3 show the iterative partitioning for the benchmark circuits being used in this dissertation. For the circuits *Ckt3* and *Cramb alone*, there are still a few subcircuits with a large number of nodes. These large subcircuits belong to special structures such as pass-transistors, memory arrays, or analog blocks which need other methods for further partition. Nevertheless, for the main objective of our research, this partitioning technique may still represent a reasonable load balance between processors, with the exception of the *Cramb* circuit.

The first two columns of Table 3 indicate the circuit name and number of subcir-

TABLE 3 Iterative circuit partitioning using $\alpha_I = 0.5$ and $\alpha_{II} = 0.8$

Ckt	# Subckts	Width	Height	Avg # Nodes	Max # Nodes	Simulation Time (sec)
<i>Decpla</i>	31	2.0	15	1.8	5	17.8
<i>Scdac</i>	60	2.85	21	2.5	7	68.8
<i>Cramb</i>	63	3.4	22	2.2	21	105.6
<i>Ckt3</i>	80	4.1	17	3.75	18	204
<i>Decpla8</i>	175	11.66	15	2.5	8	135

cuits. The next two columns indicate the nature of the subcircuit task graph after the iterative circuit partitioning operation. Each of the circuits is described in detail as follows.

Decpla is the example of a small circuit with a narrow task graph. An average width of only 2.0 makes any simulation scheme that strictly follows the task dependencies difficult to parallelize. Theoretically, subcircuit level parallelism can only achieve a speedup of less than 2.0. This circuit is a real industrial circuit (critical portion of a PLA) and captures many aspects of standard circuits in a small example.

The *Scdac* circuit graph is much wider and has a tree-like interconnection. Most of the larger subcircuits are located at the end of the tree.

The *Cramb* subcircuit graph is long and narrow. Its graph has a wider upper part and a narrow tail. In addition, there is a very large subcircuit along the tail which is about 10 times bigger than the average subcircuit. This circuit is interesting since the large subcircuit will dominate the simulation process and become the bottleneck.

Ckt3 is similar to the *Cramb* circuit except that each subcircuit for *Ckt3* is about twice as big as those in *Cramb*.

Finally *Decpla8* has eight copies of *Decpla*, therefore, the potential parallelism of this circuit can be exploited at its maximum with eight processors

4.1.2 Temporal Domain Decomposition

Partitioning in the Time Domain in WR is necessary because for some types of circuits, the relaxation converges in a "nonuniform" manner, such that the number of iterations required for convergence is proportional to the simulation interval. This situation can be observed in problems that contain global feedback, such as a ring oscillator or a finite state machine. The solution to this problem is to break up the simulation period into "windows" such that the WR iteration converges uniformly over each window. Selecting equal size windows is not the best approach to decomposing the simulation in the temporal domain because of the local and global feedback effects in a circuit. Local feedback is a determinant factor in the numerical convergence, implying many iterations if it is highly increased. This is the case of the Miller's capacitance effect when the voltage drain-source of an MOS transistor is in the linear region. Global feedback effect is also to be considered in the selection of the window size. This is the case when the window size is bigger than the delay time around the loop. The waveforms can become incorrect during the first few iterations, because convergence in WR grows in an exponentially weighted norm [74].

4.1.2.1 Fixed Time Window Decomposition

Initially on this research, a WR simulator was implemented based on the iSPICE3 simulator, which uses an event-driven ITA analysis algorithm. The program was modified to include a fixed time window scheduler for each subcircuit using the Gauss-Seidel ordering approach to obtain faster convergence. The general program structure is shown in figure . Two set of experiments were carried out. One corresponds to a chain of inverters with a very small coupling between them, by making the node

```

main () {
    read_input_netlist();
    setup_data_structures();
    partition();
    buildsubs();
    dc_solver();
    allocate_time_windows_for_subcircuits_as_events();
    start_transient();
    print_waveforms();
    print_statistics();
}

start_transient() {
    for (each_time_window) {
        while ( waveforms_not_converged) {
            for (all_subcircuits) {
                simulate (subcircuit, window, starttime);
            }
        }
    }
}

```

FIGURE 30

WR program structure

capacitance at each node larger than the feedback capacitance $C_L \gg C_f$. The other test circuit is based on the ring oscillator with the same relationship between capacitance values as the first circuit. Both circuits were simulated with a different number of inverters. Figure 31 shows the run time for a chain of N inverters ($N=20$ to $N=80$) using the WR, iSPICE3 and HSPICE. As the window size becomes larger in WR the CPU time

FIGURE 31

Run times for a chain of N inverters using WR, iSPLICE3 and HSPICE

is decreased, mainly because the subcircuits are very loosely-coupled and better multirate properties can be exploited.

Therefore, the larger the window size, the better the algorithm can exploit the multirate properties. For $N=80$ inverters, the WR is 5.5 times faster than HSPICE and 3.5 faster than iSPLICE3. For the ring oscillator, similar relative performances were obtained

which are shown in figure 32, since the coupling capacitance values were not altered. WR with $N=80$ inverters was 4 times faster than iSPLICE3 and HSPICE.

Another experiment on this simulation was to see how it would perform when the feedback loop became smaller. To achieve this, a higher threshold α value was used, until each inverter became a subcircuit. The simulation performance is shown in figure 33 for $N=4$ to $N=10$ inverters. In this case, the time window had to be made smaller to provide the same performance as HSPICE. As can be seen in this example, by making the window smaller the subcircuit integration steps are limited by the window size, whose performance becomes essentially limited by the Newton-Raphson iteration phase.

Another set of experiments has been done with this simulator using the ring oscillator example, this time increasing the coupling between subcircuits or decreasing the ratio between the load capacitances C_L and the feedback capacitance C_F : $R_C = C_L/C_F$. Three sets of these ratios were used: $R_C=1, 2$, and 10 . The window size was set to $W=T_N/100$ in all cases. The threshold partitioning parameter α was adjusted in each simulation such that each inverter became a single subcircuit. This threshold parameter α was constant for each set of the R_C ratios, independent of the number of inverters in the loop. These α values were found to be $\alpha = 0.9, 0.6$ and 0.4 respectively as shown in the figure 34. The solid lines represent the speedup ratio between HSPICE and WR, while the dotted lines represent the speedup ratios between HSPICE and iSPLICE for each R_C ratio. For both speedup ratios the subcircuits were forced to be solved by single node decomposition. This example shows that iSPLICE was slower than HSPICE when forced to be solved with a higher threshold partition parameter, while the WR was still able to obtain higher speedup with respect to HSPICE because the window size was still appropriate to obtain the minimum number of iterations to obtain convergence. But when the number of inverters inside the loop become < 10 , HSPICE performs better. This

FIGURE 32

Run times for a ring oscillator with a variable number of inverters inside the loop using WR, iSPLICE3 and HSPICE

implies that for subcircuits with shorter loops, global feedback degrades convergence in WR, making DM the best choice.

FIGURE 33

Run times for a ring oscillator with a variable number of inverters inside the loop, using WR, iSPICE3 and HSPICE

FIGURE 34

Speedup ratios between HSPICE/WR (solid lines) and HSPICE/iSPICE3 (dotted lines) using different R_C ratios

4.1.2.2 Dynamic Window Decomposition

The need for dynamic windowing was introduced in RELAX2. The idea presented in [72] is to perform the simulation in time windows of variable size. The relaxation process in a window is completed before proceeding with the next one such that the relaxed waveform in a window can be dumped into disk storage and the main memory space reused for the waveforms in a new window. The window size is dynamically cut if convergence is not achieved within a prescribed number of waveform iterations in order to deal with the problem of nonuniform convergence. This method has a disadvantage when the window size is reduced globally over the whole circuit and becomes so small that the simulation efficiency drops quickly.

The RELAX2 program determines the window size dynamically by two criteria. The first criteria is based on the maximum number of points allowed to represent a node waveform in a window. This puts an upper bound limit on the storage needed for the waveform. These waveform buffers account for most of the memory used for the simulation data. The second criteria is based on equal distribution of time points in a window. Therefore, if the waveform is changing rapidly, the window should be made smaller, otherwise the window size stays the same or gets increased up to the maximum size. One-twentieth of the simulation interval is the initial guess being used as maximum window. The CedarSim simulator uses this window selection for decomposition in the temporal domain, which is used in our research as well.

4.2 Task Granularity in a WR Distributed Environment

Basically, the task size determines the granularity of the simulation. In a WR algorithm, a subcircuit in one WR iteration and in a given time window represents the task size, which could be placed in one processor for simulation. In a distributed simulation

environment it is very important to know the ratio of metric computation to communication, which is defined as the amount of CPU time consumed by the task before a communication operation to another processor takes place. The communication cost is the elapsed time for a communicated message to arrive at another processor. This ratio expresses how much overhead is incurred per unit of computation. When the ratio is very low, it becomes unprofitable to use parallelism. When the ratio is very high, parallelism is potentially profitable. Unfortunately the task size is not uniform across the simulation in WR, and the task size can only be estimated, based on the subcircuit and window size. On the other hand, communication costs vary from different type of communication network. Therefore, communication costs can be high, for example, across a network of workstations using the TCP/IP protocol, or very low using specialized hardware like the slotted ring architecture. Each of the two platforms is used as a vehicle to test our distributed WR algorithms. In Chapter 3, it was shown that both machines have similar performance in terms of CPU power. So it is expected that the simulation speedup using both platforms will be close when the metric ratio of computation over communication is very high.

Experimental benchmark circuits were simulated in a single Sun Sparc-2 station to determine the cost of a single time point. In WR the task granularity will depend on the number of nodes inside the subcircuit and in the number of time points computed within a window. The grain size in this case is highly dependent on the partitioning problem. In table 4, single time point computations are displayed for all the benchmark circuits. This single time point simulation is more or less uniform for all the benchmark circuits, varying between 2.4 ms and 3.7ms. Therefore, if the window computation for *decpla* is on the average 4 time points, the task size is around 10 ms. The upper bound is limited by the size of the buffer waveform. If this buffer size includes 60 time points, then this upper bound window computation is around 144 ms for the same circuit. Similar tests were executed on the slotted-ring architecture, which showed an average time per point of about 5 ms for the

decpla circuit, and 4.5 ms for the *scdac* circuit. The other benchmark circuits were not able to run on the slotted-ring because of limited memory space available in the system.

Waveform passing-messages were also measured for both platforms to take into account the communication cost. In the slotted-ring this overhead was measured when the waveform was packed at the transmitted end and when it was unpacked at the receiving end. Measurements taken with a logic analyzer show variations in the communication cost depending on the communication path length between processors. The measured overhead ranges from 0.35 ms to 1.5ms, which gives an average of 0.7ms/per message in communication cost. In the Sun workstation with the ethernet network is more difficult to measure the communication cost, because we have to distinguish between synchronization messages and waveform passing messages. This was done by timing the total accumulated simulation and synchronization time under optimum load conditions in the network and counting the total number of waveform messages being transmitted during the simulation. The average communication cost is the difference between real simulation time and total accumulated time divided by the total number of messages. The communication cost for the Sun network varies between 0.65 ms and 1.3 ms, which gives a communication cost similar to the slotted-ring architecture.

TABLE 4 Average time point simulation for different benchmark circuits

Circuit	Sun Sparc-2 (ms)	Slotted-Ring (ms)
Decpla	2.4	5.18
Scdac	3.5	4.5
Cramb	3	-
Ckt3	3.7	-
Decpla8	3.1	-

Considering 4 time points per window as a lower bound and the communication cost, the metric ratio of computation over communication is 9.6 for the Sun Sparc2

workstation and 20 for the slotted-ring architecture when *decpla* circuit is simulated. For the other benchmark circuits this ratio is higher, as shown in table 5. These values represent the degree of communication overhead when these benchmark circuits are simulated using WR for these two platforms. Of course the higher these ratios, the more parallelism can be expected. Another overhead that must be included is the synchronization overhead in a distributed environment which is part of the global waveform convergence method considered in chapter VI.

TABLE 5 Lower bound computation/communication metric ratio

Circuit	Sun Sparc-2 (computation/communication)	Slotted-Ring (computation/communication)
Decpla	9.6	20
Scdac	14	18
Cramb	12	--
Ckt3	14.8	--
Decpla8	12.4	--

4.3 A Portable Parallel Simulator

A portable parallel simulator called CedarSim, developed at the University of Illinois [71], is used to evaluate the WR algorithms and performance in our distributed environment. The advantage of this portable parallel simulator is that it encompasses both code portability and performance portability. The software is equipped with different levels of relaxation methods, based on the coupling degree between subcircuits. Because this software evolved through a hierarchical architecture (the CEDAR machine) [21], distributed and shared memory paradigms are supported. A hierarchical partitioning scheme tends to produce loosely coupled subcircuits at the top level and more tightly coupled subcircuits at the lower level. The top level platform is simulated using WR methods supported through the message-passing paradigm. The Iterative Time Analysis (ITA), in

conjunction with the Direct Methods (DM) and Parallel Model Evaluation (PME), are supported using the standard shared-memory paradigm. This hierarchical hybrid simulator allows the mixing of all these algorithms based on information regarding the circuit characteristics and architectural features. The algorithm framework is illustrated in figure 35. Only the top level platform of this portable parallel simulator is used, since only WR algorithm improvement techniques for distributed simulation are under consideration. Because ITA and PME are considered medium and fine grain computations respectively compared to WR techniques, ITA or PME techniques can be included in a distributed environment as long as the required communication costs are reduced, so the metric computation/communication ratio is maintained. A special hybrid architecture that fits this environment is the CEDAR machine developed at the University of Illinois [31].

FIGURE 35

Algorithm framework on the portable parallel simulator

4.4 Summary

In this chapter, several issues are investigated and a feasibility study is done to apply parallel WR for two different distributed architectures. The main issue considered in this chapter is relative to partitioning in both spatial and temporal domains. Circuit partitioning is a very critical issue in a distributed WR simulation environment, because a trade off between parallelism and convergence must be achieved. An ideal spatial partitioning scheme is one that could break the complete circuit to be simulated in equal size subcircuits, so load balance between processors can be evenly distributed, preventing starvation between processors. This spatial domain partitioning is considered crucial for an effective speedup with a number of processors. Nevertheless, there are many structures inside of the commercial circuit examples that could not be divided, although an iterative Norton admittance partitioning method is used. New methods are required to further divide these structures, which are not considered in this study.

Temporal domain partitioning was also considered in this chapter. A WR simulator was implemented based on the iSPLICE3 simulator, which includes a fixed time window scheduler for each subcircuit using the Gauss-Seidel ordering approach to obtain faster convergence. Speedup comparison are made between WR, iSPLICE3 and HSPICE using synthetic benchmark circuits, based on the coupling degree between subcircuits. Local and global feedback effects are considered in this study. Finally, a dynamic window decomposition scheme implemented in the RELAX2 simulator is also presented.

Based on both partitioning schemes, a task granularity and a metric computation/communication ratio was determined experimentally for two different architectures, based on the benchmark circuits used for the WR environment.

Finally, a portable parallel simulator the CedarSim simulator implemented at the University of Illinois, used for our study and evaluation of our WR algorithms based on our distributed environment framework, is presented.

V

WR Parallelism in the Spatial and Temporal Domain, Scheduling and Task Allocation

This chapter mainly focuses on task parallelism and scheduling for relaxation methods. Two algorithms based on asynchronous methods are presented to exploit the functional parallelism in the spatial and iteration domain for WR methods. These are the Data-Flow and the Event-Driven scheduling techniques. Temporal domain parallelism is explained using time segment pipelining. A further issue addressed in this chapter is task allocation. An efficient static subcircuit allocator is presented to obtain maximum parallelism in both spatial and temporal domains for the slotted-ring architecture.

5.1 Subcircuit Ordering

After the circuit is partitioned, the subcircuits are interconnected using a link list to build the task dependency graph. This graph represents the relationship between the subcircuits which is generated by extracting coupling information from the partitioner, as shown in figure 36a. The next step is the ordering of the subcircuits based on the logic signal flow. Although the signals are strictly bidirectional, one direction is usually more “dominant”, meaning that the signal flow is primarily in one direction. A partial ordering is generated by considering the dominant edges in the graph, and produces a directed acyclic graph as shown in figure 36b. Ordering is the process of ranking each subcircuit which

will assist the scheduler in scheduling the subcircuits for the time domain analysis. The graph in figure 36b defines the relaxation scheme by specifying the data dependencies to be enforced within one iteration of a single Gauss-Seidel iteration. As can be seen from

FIGURE 36

Examples of Gauss-Seidel and Gauss-Jacobi partial ordering for one iteration. (a) original subcircuit graph (b) Gauss-Seidel ordering $m=3$ (c) Gauss-Jacobi ordering $m=1$

this figure, subcircuits s_1, s_2, s_3 belong to the first rank level ($m = 1$) based on the distance in terms of the directed edges from an input source. Subcircuits s_4, s_5 belong to the next rank ($m = 2$) and subcircuits s_6, s_7, s_8 correspond to the last rank level ($m = 3$). Therefore, this Gauss-Seidel graph has three ranks.

The Gauss-Jacobi method is generated by placing all the subcircuits in a single rank and ignoring the directionality of the circuit within a single iteration. This is illustrated in figure 36c, in which all subcircuits $s_1 \rightarrow s_8$ are on the same rank ($m = 1$).

If feedback loops are present in the circuit, special consideration is required for faster convergence. In some other research work, feedback loops are either broken or merged so that their associated subcircuits are put into the same subcircuit. This condensation of subcircuits generates new blocks known as strong connected components (SCC). The partitioning scheme used in the CedarSim simulator is based solely on the coupling parameters. In this case, if a feedback loop is considered to be weak, few iterations should be required for the loop to converge. Hence, the subcircuits will not be merged but the edge associated with the loop would still be considered during scheduling.

5.2 Task Parallelism and Scheduling in the Spatial Domain

By using the ranking process of subcircuits of section 5.1, it is possible to simulate several tasks in parallel in the spatial domain, where each task consists in solving each subcircuit for the same time period or window. The Gauss-Seidel WR method requires that all the inputs of a subcircuit be computed in the current WR iteration before the simulation of this subcircuit can be started. This ordering of subcircuits means that all the subcircuits of a given rank can be simulated from the moment that all the lower rank subcircuits are simulated. The circuit is then solved rank by rank, starting with the subcircuits of the first rank. Since there is no dependency between the subcircuits in a given

rank, they can be simulated in parallel. When all the ranks are simulated, a new WR iteration is started until convergence is obtained. As shown in the dependency graph of figure 36b, only subcircuits s_1, s_2 and s_3 in rank $m = 1$ can be processed in parallel. The parallel implementation of this rank scheduling requires the insertion of synchronization barriers between ranks. This method can be efficient as long as the subcircuit graph is very wide and each subcircuit is of the same size. Odent et al [44] used this algorithm with a shared memory multiprocessor system, and found, as expected, that the speed-up is large for some special circuits which have a lot of functional parallelism. The main drawback of this algorithm is that the number of active processors drops quickly with the number of iterations and drops further as the ranking increases. The problem of limited algorithmic parallelism is a result of the synchronization between the ranks. At the end of each rank a number of processors becomes idle, drastically reducing the overall speedup of the algorithm. This problem can be solved by removing the synchronization points between the different ranks. A data-flow based algorithm can remove the constraints of this bottleneck, as discussed in the following section.

5.2.1 Data-Flow Based Algorithm

In the data-flow algorithm, the task is executed as soon as possible, i.e. from the moment that all data needed to execute the task are available. In the case of circuit simulation, the data needed to execute the task are the waveforms of the input nodes of the subcircuit. A subcircuit can be simulated as soon as all the waveforms of the input nodes are available. A data-flow graph which is the same as the subcircuit graph defined earlier, is required to simulate the subcircuits. Again, this method as implemented in [44] shows its limitations in terms of its dependence on the functional parallelism of the simulated circuit. However, its performance with respect to the ranking simulation algorithm improves, mainly because the synchronization points between different subcircuit ranks are

removed. This data-flow based algorithm was modified to remove the synchronization bottleneck between the successive WR iterations. This modified algorithm, called the “overlapped WR iteration method” (OWRI) requires the following conditions for a subcircuit to be simulatable in WR iteration k :

1. The waveforms of the input nodes of the subcircuit S must be computed in the same WR iteration k .
2. The waveforms of the fanout nodes of the subcircuit S must be computed in the previous WR iteration $k - 1$.

This method can be interpreted using as many replicas of the data dependency graph as the number of WR iterations required to reach convergence. The dependencies between the different WR iterations are included by adding new edges to the graph. An edge is added from subcircuit B in WR iteration k to subcircuit A in WR iteration $k + 1$ if subcircuit B is a fanout subcircuit of subcircuit A. This task dependency graph for the overlapping WR iteration is shown for a particular graph in figure 37, assuming two iterations are needed. From this graph, six subcircuits, $s_6^k, s_7^k, s_8^k, s_1^{k+1}, s_2^{k+1}, s_3^{k+1}$ can be computed concurrently, assuming all the tasks are of the same size.

Although this method proves to be better than the two previous algorithms, the speedup is still dependent on the size and the functional parallelism in the simulated circuit and the bottleneck is now in the centralized task queue utilized in the shared memory paradigm from which events are removed. Distributed algorithms decentralized the task queue management, given more freedom for simulation where processors remove tasks from their own queue. These methods, classified as asynchronous waveform relaxation methods, can obtain higher speedup, especially when the number of processors, N_p , is

larger than the average circuit task graph, G_w . Wen et al [70] prove this observation based on the parallel event-driven WR.

FIGURE 37

Task dependency graph for the overlapped WR iteration algorithm

5.2.2 Event-Driven Waveform Relaxation.

When the width of the task graph associated with the circuit is narrow compared to the number of processors, Gauss-Jacobi or other asynchronous schemes may be desired to increase the parallelism. The event-driven mechanism is better as an asynchronous mechanism in a sense because it exploits the waveform latency at the same time. This latency exploitation can be more beneficial when used in both the time and iteration domains. In fact, partial waveform convergence exploits latency in the iteration domain. Time domain latency applies when the event-driven schemes trace the active part of the circuit based on changing sources and internal waveforms. The simulation is performed only in the active part of the circuit and simulation therefore is reduced. Successful exploitation of this

latency for non-linear relaxation using the ITA algorithm in the iSPLICE3 program was accomplished by [54], but its application to the waveform relaxation algorithm is a more complex task. It must be remembered that the subcircuits in WR exploit some form of latency by using optimum integration time steps that are independent for each subcircuit. Many variations are found in the event-driven simulators. Some schemes respect the partial ordering dependency between subcircuits within a single WR iteration. Others are completely chaotic since no dependency is enforced. Therefore, by not strictly enforcing a partial ordering of subcircuits, more processor activity is expected, increasing the parallelism. In [54] an event is considered a particular subcircuit for simulation in a given time window. A subcircuit generates an event only when any of the internal nodes inside the subcircuit has not converged for the entire window. The events generated correspond to the fanout subcircuits that are connected to the non-convergence node. A parallel event-driven simulator attempts to achieve a trade-off between parallelism and convergence, by combining a data-flow scheduling approach with an event-driven technique using a priority queue mechanism. Two separate task queues with a priority scheme are used. The Data-Flow (DF) queue has a higher priority than the Event-Driven (ED) queue. Tasks are taken from the DF queue, which maintain the subcircuit ordering of the Gauss-Seidel method. Processors will look for tasks in the ED queue only if the DF queue is empty. Tasks in the ED queue are not ready to fire but they do have tasks with at least one input with new information. This approach results in good performance when feedback is present in the circuit, which is usually the case for realistic circuits. Although this approach can exploit parallelism in the spatial and iteration domain, the method does not allow generation of events in the future, so that the temporal domain could not be exploited. This implies that the simulation of the next window cannot be started until all subcircuits have converged to the end of the current window. This method is referred in

this dissertation as the *Conservative event-driven WR algorithm*, which is explored in more detail in Chapter VI.

Another form of parallelism can be found in the other domain, the “temporal domain”. The fundamental parallelism in the time domain is based on pipeline processing, which can be beneficial for serial or chain-like structures, such as a signal processing environment, in which a fixed sequence of operations (or transforms) are required, i.e., each arrival of data may have to be Fourier transformed, multiplied by a fixed frequency, filtered, etc. This kind of computing lends itself to pipelining, as explained in the next section.

5.3 Task Parallelism and Scheduling in the Temporal Domain.

The main limitation of the parallelism in the spatial domain is the amount of functional parallelism in the circuit, which corresponds to the width of the subcircuit dependency graph. Obviously, a very narrow graph will decrease the parallelism in the space domain. The objective of the proposed approach is to gain speed by exploiting the parallelism in the other domain as well, the temporal domain, which is more effective when the inter-relationship between subcircuits is strictly linear. The mechanism used to realize temporal parallelism is “pipelining”. This concept of pipeline processing is similar to assembly lines in an industrial plant, where a pipeline of assembly stations continuously assembles separate parts along a moving conveyor. Ideally, all the assembly stations should operate at an equal processing speed, otherwise the slowest station becomes the bottleneck of the entire pipe.

Pipeline processors, whose basic structure is shown in figure 38a for a $k = 4$ -stage pipeline, behave in a similar fashion. Figure 38b represents the space-time diagram when the overlapped operations are carried out for a linear precedence relation of a set of

subtasks $\{T_1, T_2, \dots, T_n\}$. Once the pipe is full, it will output one result per clock period independent of the number of stages in the pipe. Ideally, a linear pipeline with k stages can process n subtasks in $T_k = k + (n - 1)$ clock periods, where k cycles are used to fill up the pipeline or to complete execution of the first task, and $n - 1$ cycles are needed to complete the remaining $n - 1$ tasks. In a single processor, the same number of tasks can be executed in $T_1 = n \times k$ time delay. The speedup for a k -stage linear pipeline processor is then:

$$S_k = \frac{T_1}{T_k} = \frac{n \times k}{k + (n - 1)} \quad (\text{EQ 25})$$

It should be noted that the maximum speed of $S_k \approx k$ is obtained when $n \gg k$.

FIGURE 38

(a) Basic structure of a four pipeline processor
(b) Space-time diagram depicting the overlapped operations

This approach of pipelining in the circuit simulation environment was introduced in time point pipelining (TPP) to parallelize the WR algorithm in the time domain [73]. For the sequential circuit in figure 39, which shows three subcircuits arranged in a linear dependency-graph, the TPP uses the same pipeline processing to compute time point waveforms staggered in the pipeline. The process begins when one processor starts computing the transient response of a subcircuit. Once the first time point is generated, a second processor begins computing the first time point for the second subcircuit, while the first processor computes the second time point for the first subcircuit. In the next step, a third processor is introduced to compute the first time point for the third subcircuit, and so on. This algorithm was implemented in a shared memory parallel processor, where the shared memory is used to store the iteration waveforms, as well as a queue with the list of subcircuits.

Each processor starts by taking a subcircuit from the queue, then examines its respective subcircuit's external waveforms to see if the waveform values needed to compute the next integration timestep are available. If so, the next timestep for the subcircuit is computed; otherwise, the subcircuit is returned to the queue and the processor tries again with another subcircuit from the queue. As timepoints are computed, more of the subcircuits will have the information needed to compute their own timepoints. The most difficult part of this algorithm is that it creates a great deal of overhead in having every processor search through all the subcircuits to find a subcircuit for which one time point can be computed. A modification which limits the search of this algorithm is presented in [73]. Only those subcircuits for which at least one input has more time points than it had when the subcircuit was last processed, need to be considered. These are the fanout subcircuits of a subcircuit for which a time point has been computed on a processor. After the simulation of a subcircuit, its fanout subcircuits are placed in the queue. Thus, the queue contains only the subcircuits for which it is likely that the waveform values needed to compute a

next time point will be available. However, due to the multirate integration of the subcircuits, it is not possible to compute a new time point for many of the subcircuits in the queue. The scheduling overhead is very appreciable.

FIGURE 39

A sequential linear circuit with three subcircuits

5.4 Time-Segment Pipelining

This method, adopted first by [13] and later by [44], was mainly implemented in the idea of the TPP. To limit the scheduling overhead and gain from the multirate properties, the time segment pipelining (TSPL) uses a section of the simulation interval instead of a single time point. Then the whole simulation interval is divided into a number of simulation intervals of the same size called “segments”. This idea of TSPL was combined with the OWRI to obtain parallelism in both domains, temporal and spatial respectively [44] using the data-flow based algorithm. The main objective of this adapted TSPL-OWRI method is to simulate one subcircuit concurrently in two different WR iterations at the same time. This can be seen in the example shown in figure 39.

If we assume that three time segments and three WR iterations are needed to reach convergence, then the parallelism can be seen using the unrolled task dependency graph in figure 40. Following the data-flow based algorithm, segment S_{11}^1 is solved first; after that, two time segments can be solved concurrently, S_{21}^1 and S_{12}^1 . Next time, four time segments are available to be solved in parallel, S_{31}^1 , S_{22}^1 , S_{13}^1 and S_{11}^2 , and the pipeline solving process continues following the time segment dependency between subcircuits. The computation of S_{11}^2 means that the computed segments are not only pipelined between the simulation of subcircuits in the same WR iteration, but also between the simulation of subcircuits in

different WR iterations. The task-time diagram depicting the overlapped operations is shown in figure 41. From this diagram a theoretical speedup can be computed.

To make the mathematical model simple, let us assume the following conditions:

1. The test circuit consists strictly of a chain of N_{sub} subcircuits.
2. The simulation time of a subcircuit is a constant, T_{sub} , for all subcircuits and for all WR iterations.
3. The total number of WR iterations to reach convergence is known before the simulation and is given by I .
4. The simulation time of a subcircuit in one time segment is equal for all the time segments: $\frac{T_{sub}}{N_{ts}}$, where N_{ts} is the number of time segments. With these assumptions, the total simulation time on one processor is the sum of the simulation times of all the subcircuits in all the WR iterations, resulting in the following equation:

$$T_1 = N_{sub} T_{sub} I . \quad (EQ 26)$$

With reference to figure 41, the total simulation time using TSPL is:

$$T_{TSPL} = (N_{ts} + (N_{sub} - 1) + 2(I - 1)) \times \frac{T_{sub}}{N_{ts}} . \quad (EQ 27)$$

The theoretical speedup of the TSPL methods is the ratio of the sequential execution time and the parallel execution time of equation (EQ 27):

$$S_{TSPL} = \frac{N_{sub} \times I \times N_{ts}}{(N_{ts} + (N_{sub} - 1) + 2(I - 1))} . \quad (EQ 28)$$

When the number of subcircuits is very large, equation (EQ 28) reduces to the asymptotic speedup equal to $I \times N_{ts}$.

FIGURE 40

Task dependency graph for the TSPL algorithm

The TSPL algorithm implemented by [44] is a finer grain parallelism, which gives speedup factors higher than OWRI. A further observation is that the simulation time is very dependent on the size of the time segment: there is always an optimum size for the

time segment to obtain maximum speedup. If the time segment is made too small, the efficiency of the simulation drops because the multirate property of the WR method is lost. The subcircuit simulator must limit its time step to the end of this small time segment, which can drastically increase the number of computed points.

The other factors that affect the simulation time are overhead due to communication, scheduling and synchronization cost. If the time segment is too small, a subcircuit must be copied to the subcircuit simulator for each time segment, which considerably increases the communication overhead. The scheduling and synchronization cost are very difficult to measure, but are dependent on the number of processors used during the simulation. This kind of parallelism would be ideal for a pipeline machine, assuming the subcircuits have been allocated properly. The slotted-ring architecture, which is a distributed ring architecture, can benefit from the temporal domain parallelism. The problem is how

FIGURE 41

Task-time diagram showing the available parallelism in TSPL

to allocate the subcircuits between processors so that optimum parallelism can be obtained, such that communication overhead is reduced to a minimum. This task allocation is discussed in the next section.

5.5 Task Allocation in a Ring Architecture

In a distributed parallel system the assignment problem is a fundamental aspect of distributed computing. This problem arises because many subcircuits have to be distributed over several interconnected computers so that the cost of executing the circuit simulation in a distributed environment is minimized; the cost may be time, financial cost or some other measure of resources in use. Our objective in optimizing efficiency is to minimize the running time of the simulation.

This mapping problem which arises in many environments, is a very difficult one, for which no fully satisfactory solution is known. In its more general form, the problem can be shown to belong to the NP-complete class of problems for which no efficient algorithm is known.

The proposed method offers an heuristic solution to the problem. The allocation in general can be done in two ways, dynamic or static. The dynamic allocation can be used to improve the load balancing at the cost of increased runtime overhead. The main overhead is the transfer of the subcircuit parameters, as well as the present state of the subcircuit, such as the node voltages, currents and charges for the present and several previous time-points.

The static allocation saves a great deal of time overhead, since it is done only once, after the partitioning process is finished. However, load balance between processors is a

problem, since one processor may finish all its tasks and become idle, while another processor may still have many unfinished tasks.

FIGURE 42

Task allocation graph in a distributed ring architecture

The proposed method is a static one, based on the longest path of the network. The main objective is to find the most critical path on the dependency graph of the network, and to assign each subcircuit in sequence through this path to each processor. If the number of subcircuits along this path is larger than the number of processors, the sequence of assignation is repeated again. This assignment process for a commercial circuit example *decpla* is shown in figure 42.

This assignment procedure does allocate tasks based on the priority solution of subcircuits in the temporal domain. The second step of the allocation problem is to assign tasks to optimize the parallelism in the spatial domain. Therefore, any task left in the dependency graph which belongs to the same rank is assigned to a different processor. To do this, the algorithm must also backtrack the predecessor dependent nodes, such that the assignment process is not repeated on the path for the same processor. By looking at the example in figure 42, the nodes already assigned in the first step are marked with their corresponding processor number on the side of the node. Starting from rank 1, two nodes are found unmarked. Since they don't have dependency nodes they are assigned to a different processor, as shown. On the second rank, two nodes are again found unmarked.

TABLE 6 Circuit Load distribution for a system with 4 processors

Processor number	Subcircuits Allocated
PE1	5, 8, 11, 16, 21, 24, 29, 31
PE2	3, 7, 13, 15, 18, 23, 26
PE3	2, 6, 9, 10, 17, 20, 25, 28
PE4	4, 12, 14, 19, 22, 27, 30

For the dependency nodes of node 6, we found P1, which is therefore discarded. P2 is also discarded because it has already been assigned the same rank; therefore it is assigned to the next one, which is P3. The predecessor dependency nodes of node 4 are P2

and P3: therefore they are assigned to P4, and the assignation continues with the next rank until no more nodes are found unmarked. The tasks assigned to each processor are shown in table 6. This heuristic allocation mechanism produces a reasonable balance of tasks for each processor, assuming that all tasks are of the same granularity.

This task allocation is simple, providing maximum temporal parallelism along the longest path in the circuit, and maximum spatial parallelism in each rank level. This task assignation which is used in our distributed simulation environment is fundamental for obtaining maximum speedup in our benchmark circuits.

A final issue still remains to be discussed to complete our study, which is synchronism constraint in our distributed WR simulation environment, and is addressed in the next chapters.

5.6 Conclusions

This chapter explores the parallelism in the spatial, iteration and spatial domains in WR parallel simulation. Data-Flow and Event-Driven based simulators used till now have successfully exploited parallelism in the WR environment, but these techniques are still being limited to the current window computation. Although this method proves to be very robust, still lacks aggressiveness in the temporal domain. A synchronization method is required to obtain global window circuit convergence, which is analyzed in Chapter VI. This synchronization can be a bottleneck, especially if it occurs at the end of every window computation.

An optimistic method known as the Time Warping event-driven technique, allows each processor to keep computing its window waveforms at its own pace under the assumption that no messages will be received from other processors in its past simulation

time. If this condition does not hold, the processor invalidates its computation and starts over again, taking the received message properly into account. This mechanism, called "*rollback*", is a general purpose synchronization method proposed by [25] in an asynchronous distributed discrete-event simulation environment. This technique is used in Chapter VII to synchronize subcircuits in different windows, as well as different iterations for WR circuit simulation.

Finally, an heuristic task allocation algorithm is presented for the slotted-ring architecture, which is used for subcircuit allocation in a WR distributed circuit simulation environment. This method tries to exploit parallelism in the spatial and temporal domain, giving priorities to those subcircuits along the longest path in the circuit, if time segment pipelining or similar techniques are used for subcircuit scheduling.

VI Conservative Synchronization in a WR Distributed Simulation Environment

A number of different methods have been suggested in the past for maintaining the correct simulation time sequence in a WR distributed environment. In one of these methods, a processor will not simulate ahead of time in the next window unless it is certain that all the other processors have finished with their simulation in the same current window. This method, which will be applied to the synchronization between processors during the simulation of WR, is the main topic of this chapter and is known as a conservative WR synchronization method.

6.1 A Conservative WR in a Distributed Simulation Environment

The conservative waveform relaxation scheme, subdivides the simulation period into a number of windows, and WR is applied to each window in sequence. This implies that the simulation of the next window will not be started until all the subcircuits have converged to the end of the current window. This conservative method assures that the causality principle will never be broken. Because subcircuits are distributed in different processors, two types of waveform convergency are required. One is referred to as local convergency, from which a processor search tasks from its own queue, until it becomes

empty. This convergence scheme is represented by the pseudo code shown in italic letters in figure 43.

The code presented in figure 43 is based on the Data-Flow Event-Driven scheduling mechanism implemented by [70] with priority Data-Flow queue. A Data-Flow scheduling scheme code is very similar to the one shown in figure 43, except the tasks are removed from a single queue only when all the input waveforms are ready for a new subcircuit iteration. In the DF-ED scheduling scheme, an event-driven technique is used as a supplement to the data-flow driven approach. This technique uses a priority queue strategy to obtain faster convergence of the data-flow driven approach and the amount of parallelism of the event-driven approach. The data-flow driven queue has higher priority than the event-driven queue. Processors search tasks from the DF queue first. If this queue is empty, then tasks can be removed from the ED queue. In both scheduling mechanisms, waveform node voltages are transmitted only at the end of the window simulation of a subcircuit, based on their fanouts and convergence waveform. Local circuit convergence is achieved when all the queues are empty, and the processor enters the idle state.

In this situation, the processor can be in two states, active or inactive. To keep track of all the messages that may still be in transit, acknowledgment signals are used. "Active" means that the processor has not yet received all the acknowledged signals from all the waveform messages it has sent. The processor becomes inactive only when all these acknowledgment signals have been received. A message-count variable is used to keep track of these messages. This counter is incremented every time a message is sent, and decremented when an ACK signal is received. The global convergence synchronization mechanism is based on the token ring algorithm which assigns only one processor to be the root initiator for convergence purposes. In this case, this is identified with $id=0$. In the slotted-ring, this token passing is very trivial, though not when the Sun network is

```

While alldone != TRUE {
    if (critical == TRUE) { /* Processors read new window or end of
simulation messages*/
        read_message from PEO;
        process_message;
    } else { /* Processors read waveform messages, acknowledges, or criti-
cal signals */
        read_message from communication port;
        process_message;
    }
    if (Data_FlowQ != EMPTY) {
        remove the subcircuit from the Data_FlowQ;
        Simulate (subckt);
    }
    else if (Event_DrivenQ != EMPTY) {
        remove the subcircuit from the Event_DrivenQ;
        Simulate (subckt);
    }
    else Idle (id);
}

Simulate (subckt) {
    while ((starttime+windowsize) <= stoptime) {
        GetTimeStep (subckt);
        Subckt_function (subckt);
    }
    Wavechkcon (subckt);
    Schedule_Fanouts (subckt);
}

```

FIGURE 43

Parallel WR pseudo-code

used, which is physically interconnected by a bus network, through an Ethernet local network. Therefore, a logical ring is constructed in which each processor is assigned an *id* in the ring as shown in the figure 44. When the ring is initiated, the processor with *id*=0 is

FIGURE 44

(a) Unordered group of processors in a network
(b) A logical ring constructed in software

given the token. The token circulates around the ring by passing the token message from processor k to processor $k+1$ (module the ring size) in point to point messages. When a processor acquires the token from its neighbor, it checks to see if it is attempting to enter a critical region. This critical message is originated only by the root processor once it is in the inactive mode. If the processor accepts to enter in the critical region, it obtains the token, and replies immediately to the root processor with the same critical signal.

Global convergence is guaranteed to have occurred only when the root processor has received all these critical signals back. At this moment, the root processor is responsible for computing a new window size and broadcasting its value on all processors, to begin computation within a new window. The pseudo-code for the global window synchronization is shown in figure 45.

```

Idle (id) {
    if (token_with_me) {
        read_token_message();
        if (id==0) { /* root processor only */
            if (msg_count > 0)
                send_token (processor (id+1)Mod No PEs);
            else {
                broadcast_critical
                for (i=0; i <= No PEs; i++) {
                    read_message from communication port;
                    if (msg_type == CRITICAL) msg_count--;
                }
                Newwindow;
                if (alldone == FALSE)
                    broadcast ( window size);
                else
                    broadcast (end_of_simulation);
            }
        }
    } else { /* all other processors execute this code */
        if (msg_count > 0) {
            send_token (processor (id+1)Mod No PEs);
            token_with_me =FALSE;
        }
    }
}

```

FIGURE 45

Pseudo-code for global synchronization in a distributed message passing algorithm

6.2 Critical Path Analysis in the Conservative WR Distributed Simulation

In [1], Berry and Jefferson proposed the use of critical path analysis techniques to calculate the lower bound on the execution time in a distributed simulation.

FIGURE 46

Chain of four inverters and partial ordering graph with their allocated processors

The critical path analysis technique is used here to trace the sequential execution of a chain of four inverters as shown in figure 46 for a specified input with the corresponding partial graph created from the partitioning. Thus, simulation traces of individual subcircuits are first obtained by executing the simulation program using the conservative DF-ED WR distributed simulation in a single processor. A directed graph is then constructed from the trace, in which the nodes (subcircuits) correspond to events and the edges correspond to messages between nodes (node voltage waveforms). Each node is assigned a weight corresponding to the execution time of the corresponding subcircuit. Each arc is assigned a weight corresponding to a communication time. The longest path in the graph, the critical path, represents the shortest time in which this circuit can be executed using the conservative WR algorithm. Figure 47 shows the sequence of events which is generated by a single Sun Sparc-2, with two windows. The graph represents the partial order of

events that any execution must obey in order to obey the causality constraint. This is, event e_1 must be executed before event e_2 if there is a directed path from the node e_1 to the node representing e_2 .

In order to evaluate the potential concurrency, each node and arc is labeled. The number inside the node represents the real execution time in ms, while the arc represents the interprocess communication time. For a single processor, the arcs have weight 0. The total run time for a single processor is simply the sum of all the events, in this case 550 ms. The number in square brackets besides each node represents the total minimum accumulated simulated time at the node, without breaking the causality constraint. Therefore, if we take the critical path for the graph in figure 47, shown in bold arrows, it has a weight of 400 ms. So this simulation cannot finish in less than 400 ms under the conservative execution constraint. Hence, while using 4 processors as presented it is not possible to achieve a speedup factor higher than $550/400 = 1.375$. It is also noted that the processor that contains subcircuit #4 is the one which is used less during the simulation, representing 60 ms of processor time, and it therefore may achieve a utilization of up to 15% ($60/400$). The processor that has subcircuit #3 is the one that executes the most events, representing 180 ms of processor time, and its utilization is three times higher than the slowest processor at 45% ($180/400$). The speedup factor of $550/400 = 1.375$ is an upper bound on the speedup achievable by simulating under the constraint of the conservative distributed WR simulator.

The graph shown in figure 48 represents the sequence of events generated by the parallel DF-ED WR simulator executed on 4 workstations. Each subcircuit is assigned to a different workstation. The message communication cost is represented in this case by an arc weight of 1 ms each. The critical path for this graph is now increased by the arc weights (waveform messages), and the global convergence synchronization overhead,

FIGURE 47

Sequence of events generated for a chain of four inverters in a sequential WR algorithm

represented by the gray block between the two dark horizontal lines, which is included at the end of each window. This path has now a total weight of 1308 ms. The dotted lines represents the time that the processor stays idle. Thus from figure 48, PE0 is idle between 50 ms and 91 ms. The synchronization overhead, which is around 450 ms per window using 4 workstations, varies greatly with the number of users on the network. This overhead represents a big penalty for the size of the circuit to be simulated, and this situation is worsened with an increase of the number of windows in the simulation. Simulating the same circuit but with 20 windows instead, the simulation time on 4 workstations takes 10,069 ms, which is about 7.4 times the execution with 2 windows. This synchronization overhead is drastically smaller for the slotted ring architecture, and is around one order of magnitude lower.

It should be noted that if the large overhead is removed, the simulation time is 408 ms, which is the same as the critical path obtained in figure 47, plus 8 ms corresponding to the sum of the weighted arcs over the path.

In the next section, the results of real time simulation of a benchmark circuit are presented. The testbed of the simulation environment is the slotted-ring architecture and the Sun Sparc-2 workstations.

6.3 Conservative WR Simulator Performance on Parallel Architectures

In this section, a number of experiments are presented. The first set of experiments was performed on dedicated hardware, the slotted-ring architecture. Table 7 shows simulation runtimes and speedups for a set of benchmark circuits, using the proposed heuristic task allocation with DF-ED scheduling and conservative window synchronization. The first six circuits are synthetic benchmarks to test the amount of parallelism involved in the

FIGURE 48

Sequence of events generated for a chain of four inverters in a conservative DF-ED distributed WR algorithm

partitioning. The last two circuits are real industrial circuits which as discussed in Chapter I, but both circuits have been reduced in size for simulation in the slotted-ring because of the limited memory space in the hardware. Each PE has 500K word (1 word = 32 bits) of main memory.

Fourinv is a chain of four inverters and *ringosc* is a ring oscillator made of a nand

TABLE 7 WR runtimes and speedups in the slotted-ring architecture

Circuit	# of transistors	1 PE time/speedup	2 PE time/speedup	4 PE time/speedup
<i>fourinv</i> (4 subckts)	8	3.06 s/ 1.0	2.88 s/ 1.06	2.88 s/ 1.06
<i>ringosc</i> (5 subckts)	12	6.54 s/ 1.0	6.08 s/ 1.07	5.2 s/ 1.26
<i>4ringosc</i> (4 subckts)	48	8.24 s/ 1.0	4.6 s/ 1.79	2.84 s/ 2.9
<i>8ringosc</i> (8 subckts)	96	16.96 s/ 1.0	9.36 s/ 1.81	5.48 s/ 3.09
<i>4ringosc_t</i> (20 subckts)	48	48.96 s/ 1.0	26.4 s/ 1.85	22.1 s/ 2.21
<i>8ringosc_t</i> (40 subckts)	96	95.24 s/ 1.0	50.8 s/ 1.87	27.6 s/ 3.45
<i>decpla_m</i> (24 subckts)	116	24.18 s/ 1.0	18.3 s/ 1.32	15.7 s/ 1.54
<i>scdac_m</i> (63 subckts)	260	44.6 s/ 1.0	23.18 s/ 1.92	14.4 s/ 3.1

gate with four inverters. Both circuits present no functional parallelism between subcircuits, therefore very small speedup is expected from the DF-ED scheduling mechanism. The *ringosc* circuit has higher speedup because of a increased number of iterations per window. The speedup for these benchmark circuits is better visualized in a graph as shown in figure 49.

The circuits *4ringosc* and *8ringosc* replicate the *ringosc* circuit four and eight times respectively. If each circuit replica is allocated to a different processor, then maximum parallelism can be obtained from the circuit simulator, with minimum interaction between processors. Data transfer occurs only at the end of the window simulation, where it is required to synchronize for global window convergence. *8ringosc* presents more parallelism than *4ringosc*, because now two independent subcircuits are allocated to each processor, so speedup goes from 2.9 to 3.09 with 4 processors. As the number of independent subcircuits allocated to each processor increases, speedup should increase even more.

FIGURE 49

Speedup curves versus number of PEs for the benchmark circuits used in the slotted-ring architecture

The following two circuits *4ringosc_t* and *8ringosc_t* are exactly the same as *4ringosc* and *8ringosc* respectively, except that a different threshold partitioning value was used ($\alpha = 5.5$) to decrease the task granularity of the circuits. Thus *4ringosc_t* and

8ringosc_t now contain 20 and 40 subcircuits respectively, in which each subcircuit contains a single node. Of course, both circuits will take longer to converge, since subcircuits are more tightly coupled, similar to node decomposition, and therefore simulation time increases. The speedup in *4ringosc_t* was decreased mainly because the number of waveform messages has increased.

The last two circuits *decpla_m* and *scdac_m* are the commercial circuit examples modified to fit in the memory space in the slotted-ring architecture. In both cases the speedup obtained is limited by the size of the task graph width.

Table 9 shows the results of the simulation runtimes and speedups for the same benchmark circuits as above, simulated on an Ethernet network (10 Mbps) of Sun Sparc-2 workstations. When simulation is executed on the Sun network with small circuits such as the *fourinv* and *ringosc*, the synchronization overhead will overshadow the simulation overhead. Therefore, the worst cases of parallel simulation are obtained in both cases.

TABLE 8 WR runtimes and speedups in Sun Sparc-2 workstations

Circuit	1 PE time/speedup	2 PE time/speedup	4 PE time/speedup
<i>fourinv</i> (4 subckts)	2.96 s/ 1.0	9.98 s/ 0.29	21.22 s/ 0.139
<i>ringosc</i> (5 subckts)	4.22 s/ 1.0	9.77 s/ 0.43	14.5 s/ 0.29
<i>4ringosc</i> (4 subckts)	4.93 s/ 1.0	8.61 s/ 0.57	11.8 s/ 0.41
<i>8ringosc</i> (8 subckts)	9.64 s/ 1.0	11.7 s/ 0.82	13.0 s/ 0.74
<i>4ringosc_t</i> (20 subckts)	52.7 s/ 1.0	35.8 s/ 1.47	32.8 s/ 1.61
<i>8ringosc_t</i> (40 subckts)	82.4 s/ 1.0	58.6 s/ 1.4	38.8 s/ 2.12

TABLE 8 WR runtimes and speedups in Sun Sparc-2 workstations

Circuit	1 PE time/speedup	2 PE time/speedup	4 PE time/speedup
<i>decpla_m</i> (24 subckts)	18.36 s/ 1.0	20.98 s/ 0.87	22.44 s/ 0.81
<i>scdac_m</i> (63 subckts)	42.47s/ 1.0	44 s/ 0.96	40.36 s/ 1.05

The *4ringosc* and *8ringosc* are also small circuits to be considered for speedup, since the synchronization overhead still represent the major overhead in the simulation, but because the subcircuit sizes are bigger, the simulation efficiency increases.

When the circuits *4ringosc_t* and *8ringosc_t* were simulated, a parallel speedup of 1.61 and 2.12 were obtained respectively. The increased speedup is mainly due to the fact

FIGURE 50

Speedup curves versus number of PEs for the benchmark circuits used in the Sun Sparc-2 workstations

that the number of subcircuits has increased and there is enough functional parallelism to keep most processors busy. A great percentage of time is spent in sending node waveform voltage messages between processors. Finally, in the two industrial circuits *decpla_m* and *scdac_m* the parallelism is again overshadowed with the synchronization overhead at each window. Figure 50 illustrates the speedup for these benchmark circuits when Sun Sparc-2 workstations are used in the circuit simulation.

To probe that the global convergence overhead is the major difficulty in the conservative distributed WR simulator, the simulator was timed only at that portion of the program when the processor is busy in executing the task obtained from the event queue. This time is referred as *sim_only*. Each time the process receives a waveform message a counter gets incremented to indicate the number of waveform messages received from other processors. If we add to the *sim_only* the time it takes to receive all the messages, then we should obtain the simulation plus the communication overhead. This time is

TABLE 9

WR simulation, communication and synchronization overhead in the Sun Sparc-2 workstations for the benchmark circuits

Circuit	1 PE sym_only/ sim_comm/ idle	2 PE sym_only/ sim_comm/ idle	4 PE sym_only/ sim_comm/ idle
<i>fourinv</i> (4 subckts)	2.96 s	2.16 s/ 2.4 s/ 7.56 s	1.59 s/ 2.01 s/ 19.21 s
<i>ringosc</i> (5 subckts)	4.22 s	3.55 s/ 3.89 s/ 5.88 s	2.56 s/ 3.17 s/ 11.33 s
<i>4ringosc</i> (4 subckts)	4.93 s	2.43 s/ 2.81 s/ 5.8 s	1.21 s/ 1.8 s/ 10.0 s
<i>8ringosc</i> (8 subckts)	9.64 s	4.81 s/ 5.56 s/ 6.14 s	2.45 s/ 3.59 s/ 9.41 s
<i>4ringosc_1</i> (20 subckts)	52.7 s	25.9 s/ 28.6 s/ 7.2 s	15.15 s/ 19.96 s/ 12.84 s
<i>8ringosc_1</i> (40 subckts)	82.4 s	42.7 s/ 46.5 s/ 12.1 s	23.61 s/ 30.3 s/ 8.8 s

TABLE 9

WR simulation, communication and synchronization overhead in the Sun Sparc-2 workstations for the benchmark circuits

Circuit	1 PC total / sim / com / idle	2 PCs total / sim / com / idle	4 PCs total / sim / com / idle
<i>decpla_m</i> (24 subckts)	18.36 s	11.69 s / 13.6 s / 7.38 s	7.19 s / 10.4 s / 12.04 s
<i>scdac_m</i> (63 subckts)	42.47 s	20.05 s / 23.6 s / 20.4 s	11.5 s / 18.0 s / 22.36 s

referred as *sym_com*. Finally, if we subtract from the total real simulation time the *sym_*-

FIGURE 51

Runtime contributions for different synthetic benchmark circuits

for each simulated benchmark circuit. The *idle* values are not exact values, since the simulator is running under UNIX, which is characterized as a multiuser, multitasking operating system. Therefore, if someone else's task is being executed by the CPU, the simulator task has to wait in the ready queue of the operating system to be executed.

The graphs shown in figure 51 display all the contributions to runtime for each circuit simulated in the Sun network with 4 processors. These values are displayed in bar graph fashion, to show clearly that idle time during synchronization is the big constraint of this conservative WR simulator, with the exception of *4ringosc_t* and *8ringosc_t*. The next section presents the performance of the conservative distributed WR simulator in the Sun network, but with more commercial circuit examples added.

6.4 Distributed Circuit Simulation WR Performance in the Sun Network

The distributed DF-ED WR algorithm has been tested out on the Sun Sparc-2 workstations using up to eight processors and the same heuristic task allocation algorithm proposed in chapter 5. Performance results were obtained for 5 different industrial benchmark circuits, and runtime and speedup results are summarized in table 10. *Decpla* and

TABLE 10

Experimental runtimes and speedup for commercial circuits using a distributed DF-ED WR circuit simulation in Sun-Sparc-2 workstations

Circuits	1 PE runtime/ speedup	2 PE runtime/ speedup	4 PE runtime/ speedup	8 PE runtime/ speedup
<i>Decpla</i> (31 subckts)	17.73 s / 1.00	21.9 s / 0.8	25.7 s / 0.69	26.6 s / 0.67
<i>Scdac</i> (60 subckts)	68.5 s / 1.00	42.8 s / 1.6	48.9 s / 1.4	49.5 s / 1.38
<i>Cramb</i> (63 subckts)	105.2 s / 1.00	76.5 s / 1.37	65.3 s / 1.61	58 s / 1.81

TABLE 10

Experimental runtimes and speedup for commercial circuits using a distributed DF-ED WR circuit simulation in Sun-Sparc-2 workstations

Circuits	1 PE runtime/ speedup	2 PE runtime/ speedup	4 PE runtime/ speedup	8 PE runtime/ speedup
<i>Ckt3</i> (80 subckts)	202 s / 1.00	165 s / 1.22	166.7 s / 1.21	103.6 s / 1.95
<i>Decpla8</i> (159 subckts)	143 s / 1.00	97.2 s / 1.47	77.1 s / 1.85	43.8 s / 3.26
<i>Decpla8_1</i> (8 subckts)	106.7 s / 1.00	59.9 s / 1.78	39.5 s / 2.7	28.95 s / 3.68

Scdac circuit simulation results in slightly increased performance as from the modified *decpla_m* and *scdac_m* circuits. *Cramb* and *Ckt3* are two medium size circuits in which the speedup performance is increased, but the performance degrades when the number of workstations is increased to 8. *Decpla8_1* represents 8 identical copies of the *Decpla* circuit. Therefore *Decpla8* with 8 subcircuits is based in a loosely coupled partitioning, rep-

FIGURE 52

Speedup curves versus number of PEs for commercial circuits used in the Sun Sparc-2 workstations

representing a very small interaction between processors when they are simulated, in this case producing the largest speedup. The same circuit *Decpla8* is partitioned with the same iterative partitioning scheme, creating a large number of subcircuits. Therefore, a large degree of communication is required between processors. Speedup is still reasonable, in this case maintaining a well balanced load between processors. Figure 52 illustrates these speedups for the commercial benchmark circuits using up to 8 Sun Sparc-2 workstations. Table 11 summarizes the contributions to runtime for these industrial benchmark circuits. Notice how the *idle* time gets increased with the number of processors. These contributions are better visualized in a graph as shown in figure 53.

TABLE 11 WR simulation, communication and synchronization overhead in the Sun Sparc-2 workstations for industrial benchmark circuits

Circuit	2 PE sym_only/ sim_com/ idle.	4 PE sym_only/ sim_com/ idle	8 PE sym_only/ sim_com/ idle
<i>Decpla</i> (31 subckts)	9.6 s/ 11.5 s/ 10.85 s	5.5 s/ 8.8 s/ 16.9 s	4.3 s/ 8.46 s/ 18.14 s
<i>Scdac</i> (60 subckts)	28.4 s/ 32.1 s/ 10.7 s	16.1 s/ 23.6 s/ 25.3 s	10.82 s/ 21.4 s/ 28.1 s
<i>Cramb</i> (63 subckts)	48.6 s/ 55.4 s/ 21.1 s	27 s/ 34.5 s/ 30.8 s	18.7 s/ 27.7 s/ 30.3 s
<i>Ckt3</i> (80 subckts)	121.7 s/ 135.7 s/ 29.3 s	71.6 s/ 91.8 s/ 11.4 s	34 s/ 51.9 s/ 51.7 s
<i>Decpla8</i> (159 subckts)	68 s/ 81.4 s/ 15.8 s	41.3 s/ 65.7 s/ 11.4 s	19 s/ 40.1 s/ 3.7 s
<i>Decpla8_1</i> (8 subckts)	50 s/ 56.2 s/ 3.7 s	25.1 s/ 34.2 s/ 5.3 s	12.6 s/ 24.1 s/ 4.85 s

FIGURE 53

Runtime contributions for commercial benchmark circuits

6.5 TSPL Analysis in a Distributed Circuit WR Simulation

A critical path analysis is presented here based on the time segment pipeline technique. Using the same example of figure 46, the TSPL scheme produces the sequence of events shown in figure 54. This method makes an additional speedup by dividing the time window in smaller segments. Thus, the simulation of the subcircuits in these smaller time segments can be pipelined. The time window for this example has been divided into four

FIGURE 54

Sequence of events generated by a time segment pipelining in WR with DF-ED scheduling mechanism in a distributed environment

equal size segments. Each segment in this example uses the same amount of computation of 15 ms each. The pipeline effect is illustrated in figure 54. Subcircuit #1, with only external inputs, can be simulated in the first time segment, i.e., from 0 to 15 ms. After this, the simulation of subcircuit #1 can continue in the next time segment, i.e., from 15 ms to 30 ms. The simulation of subcircuit #2 can be started for the first time segment at almost at the same time, i.e. from 16 ms to 31 ms. Note that there is a delay of 1ms to start the simulation of subcircuit #2, which is the time it takes for the first segment message to arrive from processor PE1. Due to this pipeline effect, it is possible for subcircuit #1 to start its simulation on the second iteration, while subcircuit #2 is being simulated in its last iteration. Therefore, TSPL reduces the processor idle times when circuits have narrow task graphs. Although TSPL reveals itself as a potential technique to exploit parallelism in the time domain, its effective simulation power gets reduced in a distributed simulation environment. First, the number of messages is augmented proportionally to the number of segments assigned per window, and task granularity is reduced by the amount of segment assigned per window. This means the metric ratio of computation/communication is reduced, and communication overhead becomes significant for small time segments. The second factor that limits TSPL efficiency, is that the subcircuit simulator must limit its time step to the end of the time segment. If the time segments are small this can drastically increase the number of computed time points. In this way, the multirate property, one of the main properties of the WR method, is lost.

Figure 55 shows the three dimensions of parallelism in WR. TSPL makes use of all dimensions to obtain parallelism, but its temporal domain parallelism cannot go further than the window size limit, represented by the heavy line between the windows. In other words, synchronization overhead is still present for every window with this technique. Thus the improved speedup for the circuit example of the four inverters is only $1308/1204 = 1.13$. If the synchronization overhead is removed in figure 54, the simulation plus the

FIGURE 55

Three dimensional domain parallelism in WR: spatial, temporal and iteration domains

communication time is reduced to 302 ms, which is lower than the one obtained with the conservative WR. Removing the synchronization overhead constraint for this example produces a modified improved speedup, which in this example is equal to $428/304 = 1.4$ and the real speedup is $550/304 = 1.8$.

More aggressive techniques are required to overcome the window size limit. These techniques should allow the exploration of parallelism in different windows. This means different subcircuits in different windows and in different iterations could be solved concurrently. A new technique known as time warping (TW), proposed by [25], is applied to

the distributed WR simulation environment. This optimistic distributed circuit simulation technique is presented in the next chapter, using the same kind of analysis.

6.6 Summary

A conservative WR in a distributed simulation environment is presented in this chapter. This conservative WR distributed simulator is based on a combination of the Data-Flow and the Event-Driven scheduling techniques. A token passing mechanism is used to synchronize the global window convergence.

A critical path analysis is used to trace the execution for a chain of four inverters, with one and four processors.

Experimental simulations were obtained for synthetic and commercial benchmark circuits in two different platforms, the slotted-ring architecture and the network of Sun workstations. Although both platforms have similar computation/communication metrics, the slotted-ring architecture shows maximum speedup in all cases, because the global synchronization overhead is much reduced.

Finally the TSPL was analyzed for the distributed WR simulation environment. Task granularity is one of the constraints to be ported in a distributed simulator such as the network of Sun workstations. Although this technique proved to increase parallelism in the temporal domain, its performance was overshadowed by the amount of synchronization and communication overhead imposed by the conservative distributed algorithm.

VII Time Warping in the Waveform Relaxation Distributed Circuit Simulation Environment

This chapter is dedicated to the analysis and experimentation of the Time Warping method applied to Waveform Relaxation for the acceleration of circuit simulation in a distributed environment.

In section 7.1, we discuss the main limitations of the conservative WR distributed circuit simulation method used in Chapter VI. The limitations of this method are overcome with the proposed Time Warping technique used in the distributed discrete event-driven simulation. This technique is presented in section 7.2, in the context of WR circuit simulation. Section 7.3 illustrates how TW can improve the conservative WR simulation based on a simple example with two inverters. A critical path analysis in section 7.4 shows the potential concurrence of this new method, by exploiting parallelism in all domains. The implementation of the algorithm is presented next, continuing with the results of the simulation for the benchmark circuits.

7.1 Main Limitations for Conservative Distributed WR Methods

If Gauss-Seidel ordering is strictly obeyed in a distributed event-driven circuit simulation, most of the time processors have to wait for others to finish their tasks to continue their own simulation, based on the subcircuit dependency graph created by the

partitioner. An absolute dependency between tasks is obeyed in the data-flow (DF) mechanism. The speedup for this DF method normally degrades when the number of processors is larger than the average circuit task graph width. This limitation can be alleviated using a lower relative dependency between tasks. This implies that subcircuits with at least one updated input waveform are candidates ready for simulation. This asynchronous technique known as the event-driven approach, creates more parallelism at the expense of more wasteful work. The event-driven (ED) algorithm as implemented by [70] is mainly used to exploit the waveform latency in a circuit using partial waveform convergence (PWC). In general, for large circuits, some components of the circuit converge more quickly than others. Therefore, an event-driven method is considered beneficial when it can exploit the waveform latency in the iteration domain. In fact, Wen in [71] found that when combining both DF-ED techniques an improved performance is obtained. This DF-ED technique made use of priority queues to schedule the tasks, giving higher priority to the DF queue.

In all the cases presented above, the simulation time never advanced ahead for the next window, unless all the subcircuits converged in the current simulated window. All these techniques are classified in this dissertation as conservative WR, and a conservative distributed event-driven waveform relaxation is the main topic discussed in Chapter VI.

There are two main reasons why we postulate that the conservative DF-ED technique may create more parallelism in circuit simulation if the time window barrier is broken:

1. Subcircuit tasks become scarce when the average subcircuit task dependency graph width is less than the number of processors, creating processor starvation. This situation worsens with the number of iterations, since more subcircuits have already converged, reducing the amount of work.

2. Global waveform circuit convergence in a distributed environment consumes a significant amount of simulation time. This synchronization barrier gets worse when the number of processors get larger.

To address the first concern, a strategy is proposed to add a third level of queues to the DF-ED technique, to be used for scheduling tasks for future events. The mechanism to be used is based on the Time-Warping (TW) technique implemented originally by [25]. Although Time Warping has been successfully applied to the simulation of circuits at a logic level [63], applying TW at the electrical level in Waveform Relaxation has not being explored before. In this TW technique a new queue is created to insert all the subcircuits that are available for all the future windows contained in the simulation interval. The future subcircuit events are ordered in increasing window time. The TW queue has the lower priorities of all the queues; thus, events are taken from this queue only when the other DF and ED queues are exhausted. This implies that, instead of a processor staying idle as in the conservative methods, the processor proceeds by removing tasks from the TW queue, simulating subcircuits for subsequent future windows. This results in processors able to simulate different subcircuits in different iterations and in different windows.

To overcome the second limitation, it is proposed that processors synchronize by event-messages which contain the timestamp of the current simulation. If a processor receives a timestamp from the past of its current simulation, the processor must roll back its simulation to a time equal to the timestamp received, to recover from the causality error.

Next section introduces the Time-Warping concept in the context of Waveform Relaxation circuit simulation.

7.2 The Time Warping Mechanism applied to Waveform Relaxation

The TW mechanism is an optimistic object-oriented, distributed, discrete-event simulation mechanism proposed by [25]. Objects (subcircuits) are sequential processes that execute in parallel and communicate only by exchanging event messages which are stamped with simulation times. An event message corresponds to an event notice in sequential simulation, but instead of one central event queue global to the entire simulation, each object (subcircuit) has an incoming message queue, and the collection of queues taken together constitutes a distributed event queue. Objects execute their input messages in the timestamp order, and record their local simulation times in local clock variables.

Each subcircuit has a local clock which contains its own up-to-date simulation time. In [25], this local clock is called the local clock variable, and its content the local virtual time (LVT). The local clock of a subcircuit is updated with the timestamp of each event-combination it processes. In the Time Warping - Waveform Relaxation (TW-WR) technique, subcircuit LVTs are stamped with the window simulation starting time, which may move back as well as forward; subcircuits do not know or use the simulation clocks of other subcircuits communicating with them. This asynchronous computation may create conflicts between the local clocks of communicating subcircuits. A conflict between two clocks is created when a subcircuit A has advanced its simulation time ahead of another subcircuit B, and A receives a message from B in A's simulating past. Such a message is called a *straggler*. The conflict indicates that causality among events was violated and that A has advanced its computation incorrectly since it did not process its input messages in the proper, timestamped order.

In order to maintain correct sequencing of events within a subcircuit, the TW mechanism *rolls back* part of its computation upon detection of a straggler's message. This is done by retracting the offending subcircuit to an earlier simulation time, whose

state of variables contents and local clocks are restored to a consistent state at a simulation time just as the straggler's timestamp is.

The rollback mechanism corrects the causality error within the affected subcircuit. However, it is necessary to deal with side effects of the causality error created in other subcircuits. By executing incorrect simulation, a subcircuit may have sent event messages to other subcircuits which in turn erroneously updated their own state and LVT clocks. These intermediate side effects in the form of event messages sent to other subcircuits are undone by means of *antimessages*.

Because of rollback, the processors must periodically save the state of all variables local to the subcircuits. Therefore, current, charge and node voltage waveforms need to be saved. This implies that the TW-WR may use much more memory than the conservative WR mechanism. This memory-speed trade-off is worth considering when parallelism is to be exploited in all three domains: spatial, temporal and iteration. These types of parallelism are illustrated with a few examples in the next sections.

7.2.1 Antimessages

For every event message sent, an antimessage is kept at the subcircuit's output queue. An event message and its antimessage are identical, except for their type. An event-message has a positive (+) type, and its anti-message has a negative (-) type. During rollback, anti-messages are dispatched to unsend some previously sent event-messages. These anti-messages might in turn cause other subcircuits to roll back. Rollbacks caused by anti-messages are called secondary rollbacks. This effect can be several levels deep, depending how far the simulation problem has been spread.

Therefore, stragglers can be of positive or negative type. If the straggler is of positive type, it is inserted at the proper subcircuit's input queue. On the other hand, if it is of

negative type, then both the anti-message and its positive counterpart are removed from the subcircuit's input queue. In [25], this effect is called *annihilation*. After rollback, the subcircuit resumes the simulation from its new LVT clock.

7.2.2 Aggressive Cancellation

Rollback is a two-step process of restoration followed by cancellation. The original Time Warp mechanism described in [25] performs cancellation by sending out all anti-messages right after the restoration phase of rollback and is therefore referred to as aggressive cancellation. The advantage of this method is that when messages are wrong, the subcircuit reports them wrong as soon as it can, therefore cancelling their effects as soon as possible. Rapid cancellation can prevent cascading of anti-messages, and hence rollbacks. Aggressive cancellation is inefficient in cases when stragglers have limited effect on the offended subcircuit, especially on the messages it sends.

7.2.3 Lazy Cancellation

Lazy cancellation in rollback introduces yet another dimension of optimism to the original Time Warp mechanism [15]. It performs the restoration phase of rollback just as the original Time Warp mechanism, but the cancellation phase does not follow right after the restoration phase. Antimessages are sent only when evidence has established that a previously sent event-message is not correct or should not be sent. The subcircuit waits until the timestamp of the output message has been reached again after rollback before deciding whether to cancel it. Therefore, Time Warp with lazy cancellation tends to induce less message traffic.

7.2.4 Data Structure of a Subcircuit in TW-WR

Each subcircuit in TW-WR is represented with the following data structures:

1. The event-message data structure that has four components: The subcircuit number (pid), the time that the event is generated (timestamp), the node number that generates the event (nodenum) and the type of the message (type).

2. The anti-message data structure, which contains the same records as the event-message data structure.

3. A state data structure, which stores the charge, current and node voltages of all the nodes inside the subcircuit from the past of its simulation. This structure contains all data points since the last garbage collection, which is described later. When a straggler arrives, the subcircuit must roll back to the most recent past state available which correspond to the straggler's timestamp.

4. An input message queue, which contains all of the event-messages that have been received by the subcircuit since the last garbage collection, including messages that have already been processed. The reason for keeping the processed messages is that when a rollback occurs, the messages must be reprocessed.

5. An output message queue, which contains all the event-messages the subcircuit has sent since the last garbage collection. If the subcircuit rolls back its computation, all the messages it sent to other subcircuits from the simulation time at which it rolled back until the present can be found in the output queue so that the proper antimessages can be generated.

6. A Local Virtual Time (LVT) clock, which it represents the simulation time at which the event occurs during the execution of the event. Otherwise, it contains the lowest timestamp of all unprocessed messages. When a message arrives into the subcircuit input queue, its timestamp is compared with the LVT. If the message's timestamp is less or equal to LVT, a rollback is initiated.

7.2.5 Global Virtual Time

Global virtual time (GVT) is the time of the local clock that is farthest behind for all simulating subcircuits. GVT gives the lower bound on the progress of a simulation in terms of simulation time. In [25], Jefferson defines GVT at real time t as the minimum of all LVTs at time t . Since a subcircuit cannot send a message into its past, subcircuits can never roll back to a time earlier than GVT. All messages and states saved prior to GVT are subject to *garbage collection*, which frees their storage space. One possible means of determining GVT is, at regular intervals of the entire simulation, to halt all processors so that they multicast the lowest LVTs between all processors, and the minimum of all LVT values is used as the GVT. We discuss the implementation of the GVT for the TW-WR in the next section.

7.3 Graphical Example of Two Inverters Simulation using TW-WR

The Time Warp rollback mechanism is better explained with a simple example as shown in figure 56. The circuit partitioner creates two subcircuits from the two inverters connected in a chain which are allocated to two different processors.

For this example, we assume that $C_F \ll C_L$ so that only two iterations are required to obtain waveform convergence. The total simulation time for this particular example is 100 μ sec and is divided into two equal size windows of 50 μ sec each. Therefore subcircuit #1 is the only subcircuit that has events in its input queue, which is the one connected to an external input. There are two events in its input queue as shown in figure 57a at simulated *time* = 0. One event corresponds to window W_1 with timestamp 0 and the second event corresponds to window W_2 with a timestamp of 50 μ sec.

The sequence of events generated by the TW-WR mechanism are illustrated in fig-

FIGURE 56

Two inverters subcircuits simulation using the time warping mechanism

ure 58. The small circle and square represent the beginning and ending of the real time window simulation respectively. Processor PE1 processes its first event by simulating subcircuit #1 in the first window and in its first iteration. At the end of 10 ms, PE1 has computed $v_{1,1}^1$, which is transmitted to PE2 as an event message and the event is also inserted in its own output queue (figure 57b). At 11 ms, PE2 receives this message and creates an event in its input queue to be simulated (figure 57c), which corresponds to the simulation of subcircuit #2 in the first window in the first iteration. At the same time PE1 start processing its second event, which corresponds to the simulation of subcircuit #1 at the second window and the first iteration. This effect is similar to the time segment pipelining method, but instead the complete window size is pipelined in this case. PE1 is simulating the second window, while PE2 is simulating the first window. At time 21 ms, PE1 processes its second event and inserts this event in its output queue (figure 57d) and sends $v_{1,2}^1$ to PE2. At this moment, PE1 has finished its simulation because does not have more

events in its input queue. Meanwhile PE2 is finishing its second event. At 23 ms PE2 send $v_{2,1}'$ to PE1 and in its input queue has a new event to simulate which corresponds to sub-circuit #2 at the second window and the first iteration (figure 57e). At 25 ms PE1 receives a straggler message, which corresponds to an event in the past of its LVT clock (figure 57f). Thus, PE1 rolls back its simulation time and retracts its simulation at time 0 corresponding to the simulation of subcircuit #1 at the first window and second iteration. At time 35 ms, PE2 has completed its second event and sends $v_{2,2}'$ to PE1. When PE1 finishes its event it finds a new event in its input queue, but this time does not roll back since its time stamp is the same as the next LVT clock.

Because waveform relaxation is an iterative algorithm, TW-WR does not make use of the cancellation phase at all. This is due to the fact that there is always new information to be computed at each window iteration, even if a single new data point is computed at each iteration. Thus, it is better to propagate a few data points to other subcircuits than to cancel them using anti-messages as it is implemented in the original time warping mechanism. In the case where all data points are the same as in the previous iteration, no messages are sent at all. This technique will be called TW-WR with no cancellation.

7.4 Critical Path Analysis in the TW-WR Distributed Circuit Simulation

So far we have presented in Chapter VII the use of a critical path analysis to estimate the potential speedup for the circuit simulation of four inverters under the conservative synchronization mechanism. The same analysis is again presented in this section using the same circuit example as previously. The sequence of events generated from the TW-WR distributed algorithm is illustrated in figure 59.

The importance of the Time Warping technique is that it demonstrates that the critical path can be reduced from the lower bound found in the conservative scheme. The

FIGURE 57

Message-events generated in the input and output queues for the simulation of the two inverters of figure 58

FIGURE 58

Sequence of events generated for a two inverters simulation using TW-WR mechanism

TW-WR with no cancellation takes advantage of the lookahead computation only when the data-flow and event-driven queue have been exhausted. This optimistic method assumes that synchronization is not needed until it receives proof that it was. Thus, the TW-WR will proceed with the computation of the future windows on the premise that all messages with earlier timestamps have arrived. The length of the critical path shown by dark arrows in figure 59 is only 272 ms, without considering the synchronization overhead at the end of the simulation, which is shorter than the one obtained with a conservative WR of 400 ms. This indicates that the critical path is not a lower bound with the TW-WR when no cancellation mechanism is used. This method provides an speedup of $550/272 = 2.02$. An overhead in TW-WR needs to be added to allow the Global Virtual Time computation to determine the end of simulation. This GVT implementation will be presented in the next section. Note also that the critical path with TW-WR with no cancellation is similar to the one obtained with TSPL in chapter VI. When the synchronization overhead is considered, TW-WR is the best of all approaches so far, providing an improved speedup of $1308/722 = 1.81$. The main advantages of this algorithm are the following:

First, it explores parallelism at all domains, giving first priority to the spatial domain. This implies that the TW-WR will proceed in the temporal domain only when all events in the spatial domain have been exhausted.

Second, the communication overhead is still below that of the TSPL method, since a smaller number of messages is sent, and task granularity is of the same order as the conservative WR, therefore preserving the metric computation/communication.

Third, synchronization overhead is reduced proportionally to how often the TW-WR algorithm recomputes the GVT. This implies that for the benchmark examples where the synchronization overhead overshadows the simulation time, the simulation time can be improved by this factor. Computing GVT at the end of the fourth window, the runtime for

FIGURE 59

Sequence of events generated for a chain of four inverters in TW-WR distributed circuit simulation

this example with 20 windows using TW-WR is 3,180 ms. This results in a speedup of $10,069/3180 = 3.17$ compared to the conservative WR simulation. Table 12 presents the

TABLE 12

Simulation runtimes and speedup with respect to the conservative WR for the chain of four inverters in function of # of GVT computations using 4 Sparc-2 workstations.

N° of GVT computations	Window N°	simulation runtime (ms)	Relative Speedup
20	1	10,940	0.92
10	2	5,980	1.68
7	3	3,850	2.61
5	4	3,180	3.16
3	8	1,980	5.09
2	12	1,580	6.37
1	20	980	10.27

runtimes and speedup for this example in function of the number of GVT computations and the window number using four processors.

FIGURE 60

Relative speedup performance of TW-WR with respect to the conservative WR versus the number of GVT using 4 Sparc-2 workstations.

The speedup is better illustrated in figure 60 versus the number of GVT calculations. From this graph it is clear that the speedup is higher for a small number of GVT calculations. The CPU runtime versus the number of GVT computations is illustrated in figure 61. Note the linear increase in CPU runtime using TW-WR with respect to the number of GVT computations. The horizontal line in this figure represents the CPU runtime using the conservative WR circuit simulation. The two lines ideally should cross for $GVT=20$, since a TW-WR algorithm computing GVT every window is the same as the conservative algorithm. However, the TW-WR increases its overhead by searching for tasks in the TW input queues, increasing its CPU runtime over the conservative one, resulting in a crossover at No. of GVT calculations = 18.

FIGURE 61

CPU runtime for the conservative and TW-WR versus the number of GVT calculations using 4 Sparc-2 workstations

7.5 TW-WR Implementation

The TW-WR implementation is based on an extension of the DF-ED scheduling algorithm. In the Time Warp algorithm, messages have a header file that contains the start

time of the window, which indicates the timestamp of the event message, plus the node number that generates the event, and the subcircuit number which contains the node. The rest of the message buffer contains the actual node voltages for the present window that has been computed. For every node inside the subcircuit there are two waveform buffers: one that stores the present waveform being computed and the other that stores the state waveform. The second buffer contains the entire waveform from the past before the last GVT computation. This state waveform buffer must be large for two reasons: to store enough data points so that it does not overflow before a new GVT calculation is required, and to accommodate savings of the past state of the simulation in case an event-message arrives in the past, so the past state is available to recompute from the straggler's timestamp. The size for this buffer has been selected to be four times larger than the conventional WR simulator. This buffer increase over the conservative method is the simulation power/memory requirement trade-off. The state waveform buffer stores the current, charge and voltage for every single data point computed, as seen in figure 62 for node 3 in subcircuit #1. The local clock or VLT for this subcircuit is set to the minimum of the current time window computation, which is obtained from the present waveform buffer. Since trapezoidal extrapolation is used during subcircuit evaluation, it is necessary to update the waveform from the previous two data points before computing a new window.

In the case of a subcircuit rolled back into the past, the subcircuit history is updated from the state waveform buffer using the last two previous data points of the received straggler's timestamp. If the straggler is node 4 from subcircuit #2, then the straggler message is inserted in the input event-message queue of subcircuit #1 in time stamp order, and the state waveform buffer data is invalidated by the timestamp received from the straggler; the previous two data points are transferred into the new present waveform buffer to begin a new window from the straggler's timestamp, and the subcircuit then sets its LVT to the time of the new updated state. The subcircuit then

proceeds its simulation from the current LVT, using the messages in its input queue with times greater or equal to LVT. This subcircuit continues forward in simulation time until a new straggler arrives.

FIGURE 62

Graphical data structures for the subcircuit's waveform buffers

This approach offers potentially higher speedup than conservative approaches at the expense of a larger memory requirement. The GVT calculation is the trade-off between memory and speed, depending how frequently this value is computed. A way to

```

if (myid == 0) {
    synchronize (time = 4*window_size);
    request (min(LVT));
    [LVT] = min(local LVT received) from all PEs
    broadcast [LVT];
}

```

FIGURE 63

Pseudo-code for the GVT computation

limit the amount of memory needed is to compute the GVT often. Since we have enlarged the state waveform buffer to four times usual size, the GVT calculation is done at the end

FIGURE 64

GVT calculation before and after synchronization

of every four times the window size. The GVT implementation is initiated by processor with $id=0$, as shown in pseudo-code in figure 63. The *synchronize* routine is similar as the one implemented to synchronize the global window convergence in the conservative algorithm. At that moment all other processors enter in the critical region to send to the root processor the minimum of all subcircuit LVTs. Immediately after that they must read from the root processor the global minimum as $[LVT]$. At this stage, each processor discards any waveform being stored before $[LVT]$ from the subcircuit's waveform buffer, and update the local $[LVT]$. This process is known as *garbage collection*. The GVT calculation is illustrated in figure 64 before and after the synchronization. In the next section experimental results are presented for the TW-WR algorithm in a Sun Sparc-2 workstation network.

7.6 Experimental Results

This section presents the results of the synthetic and commercial benchmark circuits simulation results using the new TW-WR algorithm. The first simulation results are based on the synthetic benchmark circuits with up to four Sun Sparc-2 workstations. Table 13 summarizes the runtime and speedup of these circuits.

Circuits *fourinv* and *ringosc* are the worst cases in the simulation. Although no speedups were obtained for these circuits, their simulation times did not degrade with the increased number of processors. Compared to the conservative WR, *fourinv* performs 6.6 better when the TW-WR algorithm is used (with 4 workstations being used in the simulation). The results for the *ringosc* circuit are only 2.11 better. The GVT computations for these circuits significantly increase the simulation time. Therefore, if we decrease the number of GVT computations, speed improvement should be obtained.

For circuits *4ringosc* and *8ringosc* speedups of 1.95 up to 2.57 are obtained. Each processor gets one and two copies respectively of a *ringosc* subcircuit. The performance

TABLE 13

TW-WR runtimes and Speedups for the synthetic benchmark circuits using Sun Sparc-2 workstations

Circuit	# of translators	1 PE time/speedup	2 PE time/speedup	4 PE time/speedup
<i>fourinv</i> (4 subckts)	8	1.67 s / 1.0	3.21 s / 0.52	3.21 s / 0.52
<i>ringosc</i> (5 subckts)	12	6.07 s / 1.0	7.24 s / 0.83	6.87 s / 0.88
<i>4ringosc</i> (4 subckts)	48	5.07 s / 1.0	3.51 s / 1.44	2.6 s / 1.95
<i>8ringosc</i> (8 subckts)	96	9.57 s / 1.0	6.19 s / 1.54	3.71 s / 2.57
<i>4ringosc_1</i> (20 subckts)	48	49.56 s / 1.0	36.4 s / 1.36	20.35 s / 2.43
<i>8ringosc_1</i> (40 subckts)	96	92.21 s / 1.0	59.28 s / 1.55	34.12 s / 2.7
<i>decpla_m</i> (24 subckts)	116	18.13 s / 1.0	14.67 s / 1.23	11.69 s / 1.55
<i>scdac_m</i> (63 subckts)	260	44.5 s / 1.0	23.75 s / 1.87	12.16 s / 3.66

of these circuits is 4.5 and 3.5 times better respectively than the conservative WR techniques, using 4 workstations.

When these circuits are partitioned with a much tighter α (partitioning threshold), more subcircuits are created which become dispersed between all processors, increasing the convergence time. Speedups for these cases are between 2.43 and 2.7. For circuits *4ringosc_1* and *8ringosc_1* performance improved over the conservative WR by only 1.9 and 1.5 respectively when 4 workstations were used.

The two modified commercial circuits *decpla_m* and *scdac_m* are much larger circuits, in which the GVT computation times are negligible with respect to the simulation time. Their speedups are limited by the task graph width in both cases. The improved performance with respect to the conservative method is 1.9 and 3.3 respectively.

FIGURE 65

Speedup curves versus number of PEs for the benchmark circuits simulated in the Sun Sparc-2 workstations using the TW-WR algorithm

A very interesting observation that can be made is that these two commercial circuits simulated in the TW-WR method on a network of workstations, get similar speedups with respect to the slotted-ring architecture in the Conservative WR. This observation corroborates that in the slotted-ring the synchronization mechanism is not a big overhead in the simulation of the conservative WR mechanism.

Figure 65 shows the speedup for these circuits versus the number of Sun Sparc-2 workstations. The runtimes for these circuits are listed in Table TABLE 14. From this table we can conclude that TW-WR drastically reduced the idle time for all of the circuit examples. The longest *idle* time corresponds to the *decpla_m* circuit using 4 workstations. These contributions can be better visualized in the bar graph of figure FIGURE 66.

FIGURE 66

Bar graph display of all the contributions for the synthetic benchmark circuits using TW-WR with 4 Sun Sparc-2 workstations

TABLE 14

WR simulation, communication and synchronization overhead in the Sun Sparc-2 workstations for the benchmark circuits

Circuit	1 PE sym_only/ sim_comm/ idle	2 PE sym_only/ sim_comm/ idle	4 PE sym_only/ sim_comm/ idle
<i>fourinv</i> (4 subckts)	1.67 s	1.53 s/ 1.66 s/ 1.55 s	1.32 s/ 1.57 s/ 1.64 s
<i>ringosc</i> (5 subckts)	6.07 s	5.8 s/ 6.6 s/ 0.64 s	3.92 s/ 5.26 s/ 1.61 s
<i>4ringosc</i> (4 subckts)	5.07 s	2.32 s/ 2.32 s/ 1.19s	1.14 s/ 1.14 s/ 1.45s

TABLE 14

WR simulation, communication and synchronization overhead in the Sun Sparc-2 workstations for the benchmark circuits

Circuit	1 PE sym_only/ sim_com/ idle	2 PE sym_only/ sim_com/ idle	4 PE sym_only/ sim_com/ idle
<i>8ringosc</i> (8 subckts)	9.57 s	4.8 s/ 4.8 s/ 1.39 s	2.45 s/ 2.45 s/ 1.26 s
<i>4ringosc_t</i> (20 subckts)	49.56 s	30.96 s/ 33.7 s/ 2.7 s	15.14 s/ 19.51 s/ 0.85 s
<i>8ringosc_t</i> (40 subckts)	92.21 s	52.34 s/ 57.23 s/ 2.05 s	25.36 s/ 33 s/ 1.2 s
<i>decpla_m</i> (24 subckts)	18.13 s	11.56 s/ 12.31 s/ 2.36 s	6.43 s/ 7.4 s/ 4 s
<i>scdac_m</i> (63 subckts)	44.5 s	20.52 s/ 21.73 s/ 2.02 s	7.48 s/ 10.68 s/ 1.48 s

The next set of results belongs to the set of commercial benchmark circuits. Table 15 presents the runtimes and speedups obtained with these circuits when they are simulated on the Sun Sparc-2 workstations using the TW-WR algorithm.

TABLE 15

Experimental runtimes and speedups for commercial circuits using a TW-WR circuit simulation in Sun-Sparc-2 workstations

Circuits	1 PE runtime/ speedup	2 PE runtime/ speedup	4 PE runtime/ speedup	8 PE runtime/ speedup
<i>Decpla</i> (31 subckts)	17.73 s / 1.00	12.29 s / 1.44	10.58 s / 1.67	9.36 s / 1.89
<i>Scdac</i> (60 subckts)	68.45 s / 1.00	33.9 s / 2.0	21.7 s / 3.15	15.5 s / 4.41
<i>Crumb</i> (63 subckts)	105 s / 1.00	51.49 s / 2.0	43.2 s / 2.43	33.5 s / 3.13
<i>Ckt3</i> (80 subckts)	202 s / 1.00	96 s / 2.1	54 s / 3.74	42.3 s / 4.77
<i>Decpla8</i> (159 subckts)	143 s / 1.00	84 s / 1.7	45.25 s / 3.16	26.76 s / 5.34
<i>Decpla8_l</i> (8 subckts)	106.7 s / 1.00	51.4 s / 2.07	26.6 s / 4.0	14.49 s / 7.36

The correspondent speed-ups are shown in figure 67 using up to 8 Sun Sparc-2 workstations under the TW-WR algorithm. The runtimes using the TW-WR algorithm in all circuit examples shows a significant improvement over the conservative WR method. The improvement is from 1.64 to 3.2. One of the smallest improvements came from the circuit example *Cramb*. The topology of this circuit task graph is characterized by having a wider upper part and a narrow tail, and at the end of the tail there is a very large subcircuit, which limits the amount of parallelism obtained for this example.

Table 16 lists the runtime contributions for the commercial benchmark circuits using the TW-WR algorithm with up to 8 Sun Sparc-2 workstations. This table shows that most of the idle time is reduced in all cases, with the exception of the *Cramb* circuit. In

FIGURE 67

Speedup curves versus the number of PEs for the commercial benchmark circuits using up to 8 Sun-Sparc-2 workstations simulated under TW-WR algorithm

this example the large subcircuit at the end of the tail of the graph produces the bottleneck in the simulation. The idle time in *Cramb* using 8 processors is about 24% of the total simulation time, which is still considered a significant overhead. Meanwhile, for the other examples the idle time is significantly reduced.

Chapter VIII explores the potential concurrency using the TW-WR under ideal conditions, and presents the relative performance with respect to the conservative WR simulator using the network of Sun workstations.

FIGURE 38

Runtime contributions for the commercial circuit benchmarks using the TW-WR simulator with 8 Sun Sparc-2 workstations

Finally, figure 68 displays all the contributions to runtime for the industrial circuit examples.

TABLE 16 WR simulation, communication and synchronization overhead in the Sun Sparc-2 workstations for commercial benchmark circuits

Circuit	2 PE sym_only/ sim_com/ idle.	4 PE sym_only/ sim_com/ idle	8 PE sym_only/ sim_com/ idle
<i>Decpla</i> (31 subckts)	11.36 s/ 12.13 s/ .16 s	7.43 s/ 9.07 s/ 1.51 s	5.88 s/ 8.63 s/ .73 s
<i>Sedac</i> (60 subckts)	32.34 s/ 33.73 s/ .16 s	17.88 s/ 21.17 s/ .53 s	10.79 s/ 15.35 s/ .15 s
<i>Cramb</i> (63 subckts)	42.27 s/ 43.36 s/ 8.13 s	29.2 s/ 31.36 s/ 11.9 s	22.41 s/ 25.5 s/ 8 s
<i>Ckt3</i> (80 subckts)	89 s/ 92 s/ 4 s	48.27 s/ 53.67 s/ .33s	33.7 s/ 40.7 s/ 1.6 s
<i>Decpla8</i> (159 subckts)	78.8 s/ 83.6 s/ .4 s	38.4 s/ 44.9 s/ .32 s	17 s/ 26.1 s/ .66 s
<i>Decpla8_1</i> (8 subckts)	49.5 s/ 49.51 s/ 1.89 s	25.86 s/ 25.86 s/ .74 s	12.85 s/ 12.85 s/ 1.63 s

7.7 Conclusions

A Time Warping simulation technique was presented in the context of waveform relaxation circuit simulation. A novel technique to simulate a circuit using waveform relaxation at the electrical level in a distributed environment, it uses TW to synchronize the simulation events between subcircuits in different processors. The circuit simulation implementation is based on a third level of queues added to the original DF-ED simulator. This TW queue contains all the event-messages that the processor can simulate for future windows in the simulation interval. This TW queue has the lowest priority of all the queues; thus, events are taken from this queue only when the other DF and ED queues are exhausted. This implies that instead of a processor staying idle waiting for other processors to converge as in the conservative methods, the processor proceeds by removing tasks

from the TW queue, simulating subcircuits for subsequent future windows. This results in processors able to simulate different subcircuits in different iterations and in different windows.

Using a four inverter circuit example, it was shown that TW-WR performs better than the critical path of the conservative WR circuit simulator. The increased performance was obtained from the fact that TW-WR explores parallelism in the temporal domain as well, and since this circuit example has no functional parallelism, simulating ahead in window time generates the window pipelining effect, thus reducing the overall simulation time along the critical path.

From the experimental results, the TW-WR circuit simulator generates a significant speedup from 1.64 to 3.2 over the Conservative WR using the network of Sun Sparc-2 workstations. It was observed that the idle time was reduced in all circuit examples, indicating that TW-WR spends more time doing simulation due to rolling back of its simulation. Another benefits of the TW-WR simulator is that it removes the synchronization barrier at the end of each window, thus reducing the synchronization overhead during the simulation.

VIII TW-WR Simulator Performance

In this chapter the TW-WR performance is presented. This simulator performance is verified using a chain of 8 inverters. Runtimes for different overhead contributions are presented in section 8.1. The reduction of the synchronization overhead being inversely proportional to the amount of GVT computation is also shown.

Similar results are presented in section 8.2 for the commercial benchmark circuits, also indicating the performance improvement obtained from the TW-WR algorithm over the conservative WR method.

Finally, section 8.3 shows the potential concurrency that it is possible to obtain with the TW-WR algorithm using communication networks with higher bandwidths.

8.1 TW-WR Simulator Performance on a Chain of 8 Inverters

In this section, an experiment is presented to show the speedup improvement of the TW-WR method over the conservative WR technique. This experiment consists of a chain of 8 inverters with a train of 4 impulses applied to the first inverter. This circuit is simulated with both WR techniques using 1, 2, 4, and 8 Sun Sparc-2 workstations.

FIGURE 69

Runtime accumulation versus window instance for the *inv8* circuit, using the TW-WR (solid line) and conservative WR (dashed line)

Figure 69 displays the accumulated time at each window instance that the processors are spending during the simulation. This time excludes the communication time and the synchronization overhead. The results of the conservative method are represented by dashed lines and are always lower than those of the TW-WR method. This indicates that processors utilizing the TW-WR technique spend less time idling, and this is the case for all circuit examples. The advantage of simulating ahead of the window times comes at the expense of the number of rollbacks that the processors must make to correct for the causality error. In this experiment the number of rollbacks is shown to decrease

with the number of processors. This value has been selected as the maximum number of rollbacks between all processors.

Figure 70 shows the total runtime (the simulated CPU time plus the communication overhead) for the same circuit example versus the number of workstations. The TW-WR simulation is represented by the solid line and the conservative method by the dashed

FIGURE 70

CPU runtime plus communication overhead versus the number of Sun Sparc-2 workstations for the chain of 8 inverters, using the TW-WR algorithm (solid line) and the conservative WR algorithm (dashed line)

line. For $N=1$, the TW-WR shows an increase over the conservative one, because of the extra overhead associated with TW-WR. The TW-WR algorithm works best when the number of processors is large. This assertion too is confirmed for $N=8$ as shown in figure 70.

Figure 71 presents the total runtime considering all the contribution (*sim + comm + idle*) to the simulation for the 8 inverter circuit example. The dashed line represents the

conservative WR method, which shows a considerable synchronization overhead for $N=4$ and up. Meanwhile, for the TW-WR technique, the total runtime decreases when N increases, as shown by a solid line.

FIGURE 71

Total runtime versus number of PEs for the circuit of 8 inverters, using the TW-WR algorithm (solid line) and the conservative WR algorithm (dashed line)

It is possible to further improve the performance of the TW-WR algorithm if the memory size is increased. GVT computation is a global variable that must be updated every so often. The less often this value is computed across the computer network, the shorter the simulation time becomes. Figure 72 shows the total CPU runtime using the TW-WR algorithm for different numbers of windows. Each line in the graph indicates the simulation time at which window instance the GVT is calculated versus the number of processors. The performance using 8 workstations increases from a 1.16 speedup for 4 windows to a 2.04 speedup for 12 windows.

Next, we present the TW-WR simulation performance with commercial circuits.

FIGURE 72

Total CPU runtime versus number of PEs, using different windows values to compute GVT, for the circuit example

8.2 TW-WR Performance for Commercial Benchmark Circuits

From the experiments performed with commercial benchmark circuits, the following results were obtained. All the experiments were performed with no other computational load on the Sun workstations. Figure 73 presents the CPU time that the processor spent doing simulation only for all six benchmark circuits versus the number of workstations, using both algorithms. These graphs exclude the communication and synchronization overheads. We can observe that the usage of TW-WR results in higher processor utilization compared to the conservative WR algorithm.

Figure 74 shows the CPU runtime for the simulation of all benchmark circuits, including the simulation time and communication overheads. The main observation is that TW-WR outperforms the conservative WR simulation in all circuit examples, with the

exception of the *Decpla* circuit, but the gap between the two algorithms is much closer than when communication overhead is not considered. The simulation time in figure 74 considering the Conservative WR algorithm (segmented lines) represents the potential CPU time predicted using critical path analysis. Therefore, TW-WR can achieve higher speedups than this lower bound predicted by critical path analysis as shown in figure 73. This assumption is based on that there is no synchronization barrier at the end of each window.

If we consider this synchronization barrier, then better performance of the TW-WR is expected. Figure 75 shows the total CPU runtime, considering the communication and synchronization overheads versus the number of workstations, for all the circuit examples. From this figure we can conclude that TW-WR offers a significant CPU simulation time reduction for all circuit examples, based on the natural window lookahead feature of the WR algorithm. These tasks are only penalized when an event-message arrives in the past of its simulation known as a *straggler*. To restore this causality error, the processor must *roll back* its simulation to the time of event-message timestamp.

Figure 76 displays the number of rollbacks versus the number of workstations during the simulation of the commercial benchmark circuits. The main observation from this figure is that the number of rollbacks decreases with the increase of workstations for the simulation, since processors have fewer subcircuits to rollback. The other observation relates to the amount of feedback present in the simulated circuit. *Ckt3* is the benchmark circuit with the highest signal feedback paths in the task graph of all medium size circuits (*Scdac*, *Crumb* and *Ckt3*). As a result, subcircuits in *Ckt3* are rolled back more often during the simulation, which is identified as such in the figure 76. *Crumb* benchmark circuit is one without global signal feedback path, which is shown in figure 76 with the lower number of rollbacks.

FIGURE 73

CPU runtime versus number of PEs, considering CPU simulation only, using the TW-WR algorithm (solid lines) and the conservative WR (dashed lines) for all commercial circuits

FIGURE 74

CPU runtime versus number of PEs, considering CPU simulation and communication times, using TW-WR algorithm (solid lines) and the conservative WR algorithm (dashed lines) for all the commercial benchmark circuits

3

of/de

3

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

FIGURE 75

Total CPU runtime versus number of PEs using the TW-WR algorithm (solid lines) and the conservative WR algorithm (dashed lines) for all the commercial benchmark circuits

FIGURE 76

Number of rollbacks versus the number of PEs, using the TW-WR algorithm for the commercial benchmark circuits

The circuit which benefits the most with the TW-WR algorithm is *Decpla8_1*, since each entire copy of *Decpla* is assigned to each workstation, and since the subcircuits never interact during the simulation no rollbacks are necessary. This circuit is shown in figure 76 with an horizontal line of zero rollback.

The overall performance of the TW-WR algorithm is visualized in figure 77, where speedup curves are presented for both algorithms for all the commercial benchmark circuits. The speedup for the *Decpla* circuit is as expected, since it is the circuit which has the narrow average graph width of 2.0. The same situation exist with *Cramb* circuit, which has a large subcircuit which dominates most of the simulation time, causing significant load unbalance. The highest speedup of 7.36 obtained is for *Decpla8_1* which corresponds to results obtained in an ideal case in which subcircuits are completely decoupled from each other.

FIGURE 77

Performance improvement of the TW-WR over the conservative WR in a network of Sun workstations for the commercial benchmark circuits

FIGURE 78

Improved speedup curves of the TW-WR algorithm over the conservative WR using a network of Sun Sparc-2 workstations

The speedup improvement of the TW-WR algorithm over the Conservative WR algorithm is shown in figure 78 for all the commercial circuits. Note that the speedup improvement ranges from 1.64 to 3.2. Figure 79 shows the CPU simulation runtime versus the number of PEs for all the circuit examples. Solid lines represent the runtimes for the TW-WR simulation and the dashed lines represent the runtimes for the conservative WR simulations. This speedup and runtime improvement of the TW-WR is sacrificed by a larger memory requirement for the circuit simulation in the Time Warping environment. Figure 80 shows the amount of memory required to simulate each of these commercial circuits using the TW-WR algorithm (solid lines) and the conservative WR algorithm (dashed lines). In all circuit examples the amount of memory more than doubles using the TW-WR algorithm. This is the classical case of memory-speed trade-off.

Next, we present the potential concurrency of this TW-WR algorithm in other platforms which have faster communication.

FIGURE 79

CPU runtimes versus the number of PEs for the commercial benchmark circuits using the conservative WR (dashed lines) and the TW-WR algorithm (solid line)

FIGURE 80

Amount of memory used by the TW-WR simulator (solid lines) and the conventional WR simulator (dashed lines) versus the number of PEs, for all the commercial benchmark circuits

8.3 Potential Concurrency for Circuit Examples using TW-WR

With today's technology it is possible to build faster communication networks, increasing the communication bandwidth. Today computers can be linked via a high-speed network utilizing fiber-optic links and crossbar switches, resulting in an increase of communication bandwidth of about 100 Mbits/sec per link [4]. This increase of communication bandwidth can therefore reduce the communication and synchronization overheads during the simulation of circuits in a distributed simulation environment, reducing the TW-WR execution time even more.

In the ideal case in which communication delay between processors is zero, the upper bound of performance can be obtained from the TW-WR simulator. From the example of the chain of 8 inverters, speedup over 7 using 8 workstations is possible, as shown in figure 69. For the commercial circuit examples, similar increases in performance can be obtained, and these upper bound speedups are shown in figure 81. These values were obtained from figure 73 assuming no communication and synchronization overheads.

FIGURE 81

Upper bound speedups for the commercial circuits assuming zero communication overhead for the TW-WR algorithm

8.4 Conclusions

The TW-WR algorithms show increased performance compared with the conservative WR algorithm. This assertion is based on the critical path analysis presented for both algorithms in Chapters VI and VII. This conclusion was also confirmed from the measurements taken from real-time simulations using industrial circuit examples. The speedup improvement of the TW-WR algorithm over the conservative one was measured at between 1.64 and 3.2.

TW-WR algorithm can further increase the parallelism in circuits with narrow task graphs, exploiting temporal domain parallelism, based on time window pipelining. This was confirmed with the simulation of a chain of 8 inverters.

The TW-WR algorithm works best for a large number of processors, and the optimum time simulation performance is obtained when each subcircuit is simulated by a single separate processor.

Although the performance study of the TW-WR algorithm has been based on a network of Sun workstations, other platforms can benefit from this algorithm, compared to conservative techniques, based on the fact that parallelism can go further than the functional parallelism obtained from the subcircuit task dependency graph.

IX Conclusions

The objective of this thesis was to determine the performance of the TW-WR algorithm in a distributed simulation environment. The Time Warping algorithm implemented originally by [25] was adapted to the waveform relaxation to accelerate conservative WR circuit simulation in a distributed paradigm.

The bottlenecks of the conservative distributed WR algorithm were identified first, based on measurements and analysis taken on synthetic benchmark circuits. The communication and synchronization overheads on a network of workstations are more severe than the dedicated hardware, the slotted ring architecture, due to the limited communication bandwidth. On the other hand, in the conservative WR algorithm the parallelism is contained within the boundaries of each window. Therefore, only spatial and iteration domain parallelism can be exploited.

The Time Warping technique was introduced in the conservative DF-ED algorithm to remove the window boundaries. The resulting optimistic and aggressive simulator can exploit parallelism in all domains, temporal parallelism only being exploited when the processor has exhausted the tasks in the DF and ED queues.

The results of the experiments using commercial benchmark circuits prove that the new TW-WR algorithm performs better than the conservative WR algorithm. Memory size increase is the price paid to improve performance. Increased performance ranged from 1.6 to 3.2 when a network of Sun Sparc-2 workstations was used with a 10 Mbit/s

intercommunication link. The memory requirements to simulate the commercial circuits using the TW-WR more than doubled in all cases in comparison with the conservative algorithms.

Among overheads considered for the simulation of commercial circuits, the drop in the synchronization overheads was the most significant one. In fact, the speedup obtained from the simulation of commercial circuit examples with TW-WR on a network of workstations was similar to that obtained on dedicated hardware (slotted-ring architecture) that used the conservative WR algorithm.

It was found from the experiments that by increasing the number of processors, the TW-WR execution time can always be reduced. This is also the case when the performance of the TW-WR algorithm is maximum compared with the conservative WR algorithm.

By reducing communication delays between workstations using higher communication bandwidths, the TW-WR simulation time can be reduced further. Due to memory size constraints in the slotted ring hardware, it was not possible to implement the TW-WR algorithm. Nevertheless, based on the number of messages each processor received during the simulation with the Sun workstations, we can predict its performance. The upper bound speedup was obtained for the commercial circuit examples, with results obtained from this analysis of a chain with 8 inverters demonstrating that TW-WR can yield a speedup of 7.

The increased speedup of circuits with no functional parallelism, like the chain of inverters, is a clear indication that TW-WR exploits temporal parallelism as well, performing better than the conservative method. This analysis did not consider the synchronization overheads, and was based on critical path analysis.

9.1 Future Directions

Although the TW-WR algorithm resulted in significant speedups for all the commercial circuit examples, simulation of a circuit like the *Cramb*, which contains a large subcircuit at the end of the tail of the subcircuit task dependency graph, yielded lower speedup. There are several ways to increase the performance for this class of structures which need to be studied with more depth. New partitioning and allocation algorithms could be investigated to further improve the performance of the TW-WR algorithm for this class of circuits.

References

- [1] O. Berry, and D. R. Jefferson, "Critical Path Analysis of Distributed Simulation." Proceedings of the Conference on Distributed Simulations 1985, January 24-26, 1985.
- [2] G. Bichoff, S. Greenberg, "CAYENNE: A parallel implementation of the circuit simulator SPICE", Proc. Int. Conf. on Computer-Aided Design, Santa Clara, California, pp. 182-185, November 1986.
- [3] R.K. Brayton, F.G. Gustavson, G.D. Hachtel. "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Backward-Differential Formulas", Proc. IEEE. Vol. 60, No. 1, pp. 98-108, January-1972.
- [4] J. V. Briner, J. L. Ellis, and G. Kedem, "Breaking the Barrier of Parallel Simulation of Digital Systems", 28th ACM/IEEE Design Automation Conference, pp. 223-226, 1991.
- [5] Chien-Chih Chen, "Parallel LU Factorization for Circuit Simulation on a MIMD Computer", Proc. Int. Conf. on Computer-Aided Design, pp. 129-132, 1988.
- [6] L. Chua and P. Lin, Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.
- [7] E. Cohen, "Performance limits of integrated circuit simulation on a dedicated mini-computer system", ERL Memo. UCB/ERL M81/29, May 22, 1981.

- [8] P. Cox, R. Burch, B. Epler, "Circuit Partitioning for Parallel Processing", Proc. Int. Conf. on Computer-Aided Design, Santa Clara, CA, November 1986, pp.186-189.
- [9] P. F. Cox, R.G. Burch, D.E. Hocevar, P. Yang, B.D. Epler, "Direct Circuit Simulation Algorithms for Parallel Processing", IEEE Trans. on Comp. Aided Design, Vol. 10, No. 6, June 1991.
- [10] J. T. Deutsch, A. R. Newton , "A Multiprocessor Implementation of Relaxation-Based Electrical Circuit Simulation", Proc. 21st Design Automation Conference, 1984.
- [11] J. T. Deutsch, "Algorithms and Architecture for Multiprocessor Based Circuit Simulation", Univ. of California , Berkeley, Memo No. ERL M85/39, May 1985.
- [12] D. Dumlugol, P. Odent, J. Cockx, H. de Man, " The segmented waveform relaxation method for mixed-mode switch electrical simulation of digital MOS VLSI circuits and its hardware acceleration on parallel computers", 1986, pp. 84-87.
- [13] D. Dumlugol, P. Odent, J. Cockx, H. de Man, " Switch-Electrical Segmented Waveform Relaxation for digital MOS VLSI and its acceleration on parallel computers", IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No 6, November 1987, pp. 992-1005.
- [14] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", Proc. 19th Design Automation Conference, Las Vegas, June 1982, pp. 175-181.
- [15] M.J. Flynn, "Some Computer Organization and their Effectiveness", IEEE Trans. on Computers, C-21, No. 9, September 1972. A. Gafni, O. Berry, and D. Jefferson, "Optimized Virtual Time Synchronization," Computer Performance and Reliability, 1988, pp. 229-244.

- [16] A. Gafni, O. Berry, and D. Jefferson, "Optimized Virtual Time Synchronization," *Computer Performance and Reliability*, 1988, pp. 229-244.
- [17] K. A. Gallivan, P. Koss, S. Lo, R. A. Saleh, "A comparison of parallel relaxation based circuit simulation techniques", *Proc. ELECTRO'88 Meeting*, Boston, MA, May 1988.
- [18] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, NJ, 1971.
- [19] G. Hachtel, A. Sangiovanni-Vincentelli, "A survey of Third-Generation Simulation Techniques.", *Proceedings of the IEEE*, VOL. 69, No. 10, October 1981, pp. 1264-1981.
- [20] G. Hung, Y. Wen, K. Gallivan, R. Saleh, "Parallel Circuit Simulation Using Hierarchical Relaxation", *27th ACM/IEEE Design Automation Conference*, 1990.
- [21] G. Hung, Y. Wen, K. Gallivan, R. Saleh, "Improving the Performance of Parallel Relaxation-Based Circuit Simulators", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 11, November 1993, pp. 1762-1774.
- [22] G. Hung, "Parallel Circuit Simulation Techniques for a Hierarchical Machine", Ph. D. Thesis, University of Illinois, Urbana-Champaign, 1993.
- [23] G. Jacob, A.R. Newton, D. Peterson, "Direct-Method Circuit Simulation using Multiprocessors.", *IEEE International Symposium on Circuits and Systems Proceedings*, San Jose, CA., May 1986, pp. 170-173.
- [24] G. Jacob, A. R. Newton, D. Pederson, "An empirical analysis of the performance of a multiprocessor-based circuit simulator", *23th Design Automation Conference*, 1986.

- [25] D. Jefferson, "Fast Concurrent Simulation using the Time Warp Mechanism", 1985 Society for Computer Simulation Multiconference, San Diego, California, January 1985.
- [26] T. A. Johnson and D. J. Zukowski, "Waveform-relaxation-based circuit simulation on the VICTOR V256 parallel processor," IBM J. Res. Dev., vol. 35, no. 5/6, September/November 1991.
- [27] T. Johnson and A. Ruehli, "Parallel Waveform Relaxation of Circuits with Global Feedback Loops", 29th ACM/IEEE Design Automation Conference, 1992.
- [28] Y.H. Jun, C.W. Lee, K.J. Lee, S.B. Park, "Timing Simulator by Waveform Relaxation considering the Feedback Effect", IEE Proceedings, Vol. 136, Pt. G, No. 1, February 1989.
- [29] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", Bell System Technical Journal, Vol. 49, No. 2, pp. 291-307, February 1970.
- [30] B. Klaassen, "A Hierarchical Multirate Method for Circuit Simulation on Parallel Computers", Advances in Computing and Information - ICCI'90, International Conference Proceedings, Niagara Falls, Ontario, Canada, May 23-28, 1990, pp. 493-496.
- [31] D. Kuck, E. Davidson, D. Lawrie, A. Sameh, "Parallel Supercomputing Today and the Cedar Approach.", Science, February 1986.
- [32] B. Kruatrachue, T. Lewis, "Grain-size determination for Parallel Processing", IEEE Software, VOL. 5, No. 1, pp. 23-32, Jan. 1988.
- [33] K.S. Kundert, "Sparse Matrix Techniques," Circuit Analysis, Simulation and Design. Vol. 1., edited by A.E. Ruehli, North-Holland, 1986.

- [34] H. T. Kung, "Synchronized and Asynchronous Parallel Algorithms for Multiprocessors", in *Algorithms and Complexity: New Directions and Recent Results*, J.F. Traub, ed., Academic Press, New York, 1976.
- [35] E. Lelarasmee, A. Ruehli, A. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, VOL. CAD-1, No. 3, July 1982, pp.131-144.
- [36] D. Lewis, "Design considerations in an advanced circuit simulation accelerator", *Int. Workshop on Hardware Accelerators*, Univ. of Oxford 1987, pp. 174-186.
- [37] G. Marong and A. Sangiovanni-Vincentelli, "Waveform Relaxation and Dynamic Partitioning for the Transient Simulation of Large Scale Bipolar Circuits", *Proc. Int. Symp. on Circ. and Syst.*, pp. 32-34, May 1985.
- [38] W. Nagel "SPICE2 : A Computer Program to Simulate Semiconductor Circuits", *Electronic Res. Lab. Rep. ERL-M520*, Univ. of California, Berkeley, May 1975.
- [39] A. R. Newton, "The simulation of large scale integrated circuits", *Vol. CAS-26*, pp. 741-749, September 1979.
- [40] A. R. Newton, A. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation", *IEEE Transactions on Electron Devices*, VOL. ED-30, No. 9, September 1983, pp. 1184-1207.
- [41] M. Nishigaki, N. Tanaka and H. Asai, "Hierarchical Decomposition System and its Availability for Network Solution" 1991 *IEEE International Symposium on Circuits and Systems*, pp. 884-887, Vol. 2..

- [42] P. Odent, D. Dumlugol, H. de Man, "Hardware acceleration of circuit simulation on a multiprocessor system", Int. Workshop on Hardware Accelerators, Univ. of Oxford, 1987. pp. 154-163.
- [43] P. Odent, L. Claesen, H. de Man, "Feedback loops and large subcircuits in the multiprocessor implementation of a relaxation based circuit simulator.", 26th ACM/IEEE Design Automation Conference, 1989, pp. 25-30.
- [44] P. Odent, L. Claesen, H. de Man, "New Parallel Techniques for Simulating MOS Circuits with Waveform Relaxation Algorithms in CSWAN" in Proc. Int. Symp. on Circuit and Systems, May 1989, pp. 1166-1169.
- [45] P. Odent, L. Claesen, H. de Man, "Acceleration of Relaxation-Based Circuit Simulation Using a Multiprocessor System", IEEE Trans. on Comp. Aided Design, Vol. 9, No. 10, October 1990. pp. 1063-1072.
- [46] J. M. Ortega, W. C. Rheinboldt, "Iterative Solution of Nonlinear Equations in Several Variables", Academic Press, 1970.
- [47] D. Overhauser and I. Hajj, "Multi-Level Circuit Partitioning for Switch-Level Timing Simulation", ISCAS'88.
- [48] V. Pan, J. Reif, "Efficient Parallel Solution of Linear Systems", Proceedings of the 17th Annual ACM Symposium on theory of Computing, Providence, RI, May 1985.
- [49] C. R. Pon, R. Saleh, T. Kwasniewski, "Distributed Circuit Simulation using Waveform Relaxation in a Slotted-Ring Architecture," 1994 Canadian Conference on Electrical and Computer Engineering, September 25-28, Halifax, Canada.
- [50] N. B. Rabbat, A.L. Sangiovanni-Vincentelli, and H.Y. Hsieh, "A Multi-level Newton Algorithm with Macromodelling and Latency for the Analysis of Large-Scale

- Nonlinear Circuits in the Time Domain," IEEE Trans. on Circuits and Systems, Vol. CAS-26, September 1979, pp. 733-741.
- [51] P. Sadayappan, "Circuit Simulation on a Multiprocessor ", IEEE 1987 Custom Integrated Circuits Conference, pp. 124-128.
- [52] R. Saleh, J. Kleckner and A. Newton, "Iterated Timing Analysis and SPLICE1," Proc. Int. Conf. on Computer-Aided Design, Santa Clara, California, September 1983.
- [53] R. Saleh, "Iterated Timing Analysis and SPLICE1," Memorandum No. UCB/ERL M84/2, U. of California, Berkeley, January 1984.
- [54] R. A. Saleh, "Nonlinear Relaxation Algorithms for Circuit Simulation", Ph.D. thesis, University of California, Berkeley, 1987.
- [55] R. A. Saleh, D. Webber, E. Xia, A. Sangiovanni-Vincentelli, "Parallel Waveform-Newton algorithms for circuit simulation", Proc. Int. Conf. Comp. Design, 1987, pp. 660-663.
- [56] R. A. Saleh, K. Gallivan, M. Chang, I. Hajj, D. Smart and T. Trick, "Parallel Circuit Simulation on Supercomputers", Proc. of the IEEE, Vol. 77, No. 12, December 1989.
- [57] R. A. Saleh and J. White, "Acceleration Relaxation Algorithms for Circuit Simulation Using Waveform-Newton and Step-Size Refinement", IEEE Transactions on Computer Aided Design, Vol. 9, No. 9, September 1990, pp. 951-958.
- [58] R. Saleh, /Private Communication/, December 1992.
- [59] A. Sangiovanni-Vincentelli, L.K. Chen and L.O. Chua, "A New Tearing Approach-Node Tearing Nodal Analysis", Proc. IEEE Int. Symp. on Circ. and Sys., 1977.

- [60] A. Sangiovanni-Vincentelli, "Parallel Processing for Simulation of VLSI Cicuits",
Invited paper in *Parallel Processing and Applications*, Elsevier Science Publisher B.V.
(North-Holland), 1988, pp. 3-19.
- [61] P. Saviz and O.Wing, "PYRAMID - A Hierarchical Waveform Relaxation-Based Circuit Simulation Program", *Proc. Int. Conf. on Computer-Aided Design*, pp. 442-446,
Nov. 1988.
- [62] P. Saviz and O. Wing, "Dynamic Circuit Restructuring for Hierarchical Waveform Relaxation", 1990 IEEE International Symposium on Circuits and Systems, pp. 519-522.
- [63] L. Soule, T. Blank, "Parallel Logic Simulation on General Purpose Machines", 25th ACM/IEEE Design Automation Conference, 1988, pp. 166-171.
- [64] H. S. Stone, "High Performance Computer Architecture", Addison-Wesley, Reading, MA, 1987.
- [65] D. Tabak, "Logarithmic Indices for Multiprocessor Evaluation", *Computer Architecture News*, VOL. 16, No. 1, pp. 85-90, March 1988.
- [66] R. Tarjan, "Depth-First search and linear graph algorithms ", *SIAM J. Comput.*, Vol. 1, No. 2.
- [67] R.S. Varga, *Matrix Iterative Analysis*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1962.
- [68] D. Webber and A. Sangiovanni-Vicentelli, "Circuit Simulation on the Connection Machine ", *Proc. 1987 Design Automation Conference*, Miami Beach, July 1987.
- [69] W.T. Weeks, A.J. Jimenez, G.H. Mahoney, D. Mehta, H. Qassemzadeh, and T.R. Scott, "Algorithms for ASTAP - A Network Analysis Program," *IEEE Trans. on Circuit Theory*, VOL. CT-20, No 6, November 1973, pp. 628-634.

- [70] Yen-Cheng Wen, K. Gallivan, R. Saleh, "Parallel Event-Driven Waveform Relaxation", Proc. ICCD'91, Boston, October 1991.
- [71] Yen-Cheng Wen, "Portable Relaxation-Based Circuit Simulation on MIMD machines", Ph. D. Thesis, University of Illinois at Urbana-Champaign, 1994.
- [72] J. White and A. Sangiovanni-Vincentelli, "RELAX2: A New Waveform Relaxation Approach for the Analysis of LSI MOS Circuits", Proc. Int. Symp. on Circ. and Sys., Newport Beach, May 1983.
- [73] J. White and A. Sangiovanni-Vincentelli, "Partitioning algorithms and parallel implementations of waveform relaxation algorithms for circuit simulation", in Proc. Int. Symp. on Circuit and Systems, 1985, pp. 221-224.
- [74] J. White and A. Sangiovanni-Vincentelli, "Relaxation Techniques for the Simulation of VLSI Circuits", Kluwer Academic Publishers, Boston, MA. 1986.
- [75] J. White, "Parallelizing circuit simulation - A combined algorithmic and specialized hardware approach," Proc. IEEE Int. Conf. Comp. and Design, NY, October 1985.
- [76] O. Wing and J. Huang, "A Computation Model of Parallel Solution of Linear Equations", IEEE Trans. Comput., Vol. C-29, pp. 632-638, 1980.
- [77] Gung-Chung Yang, "PARASPICE: A Parallel Circuit Simulator for Shared-Memory Multiprocessors", 27th ACM/IEEE Design Automation Conference, 1990.
- [78] P. Yang, I. Hajj, and T. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing," Proc. Int. Conf. on Circuits and Computers, New York, October 1980.
- [79] G. Yaremchuk, "Design and Implementation of a Multiprocessor for DSP and Communication Systems Algorithm Evaluation", M. Eng. Thesis, May 1993, Carleton University, Ottawa, Canada.

- [80] G. Yaremchuk, C. R. Pon, T. Kwasniewski, R. Goubran, "A Novel Slotted-Ring Architecture for Parallel Processing: An Application," 1994 Canadian Conference on Electrical and Computer Engineering, September 25-28, Halifax, Canada.
- [81] H. Yoshida, S. Kumagai, I. Shirakawa, "A parallel implementation of large-scale circuit simulation", Int. Workshop on Hardware Accelerators, Univ. of Oxford, 1987.
- [82] Ch. Yuan, R. Lucas, Ph. Chan, R. Dutton, "Parallel Electronic Circuit Simulation on the iPSC System", IEEE Custom Integrated Circuit Conference, 1988.

Appendix A: Solution Steps of Circuit Simulation using DM

The following sections explain the stages of standard circuit simulation in some detail.

A.1 The Integration Algorithm

The goal of an integration algorithm is to model a continuous-time system by a discrete-time system. Therefore, to maximize the computational efficiency of a given integration algorithm, it is desirable to minimize the number of time-steps by using the largest possible step size.

In general, circuit equations tend to be stiff, meaning that the natural frequencies (time constants) of the circuit are widely separated, requiring a stiffly stable implicit discretization function for stability reasons. Two of the most common implicit discretization functions used in circuit simulation are the second-order Trapezoidal Rule and the variable-order Backward Differentiation Formula (BDF).

Applying the trapezoidal integration formula to (EQ 1) in Chapter II, for a time step h , yields

$$q(v(t+h), u(t+h)) - q(v(t), u(t))$$

$$= 0.5h [g(v(t+h), u(t+h)) + g(v(t), u(t))] \quad (\text{EQ A.1})$$

where $v(t)$ is known, and the equation (EQ A.1), must be solved to compute $v(t+h)$. The step size h is selected automatically at each time step to keep the discretization error sufficiently small, and to try to keep the minimum number of points. The equation (EQ A.1) has now become a set of implicit non-linear algebraic equations, which are solved by the iterative Newton-Raphson algorithm.

A.2 Newton-Raphson Linearization

For solving a general nonlinear system of the form

$$F(x) = 0; \quad (\text{EQ A.2})$$

a Newton-Raphson method shows that the solution is found by iteratively solving the following linear algebraic equation:

$$J_F(x^k) (x^{k+1} - x^k) = -F(x^k) \quad (\text{EQ A.3})$$

where (x^k) represents the value of x at iteration k , J_F is the Jacobian matrix of F with respect to x , and $F(x^k)$, the right-hand side term, is referred to as the Newton residue.

If the Newton algorithm is used to solve equation (EQ A.1) for $v(t+h)$, the residue $F(v^k(t+h))$, is

$$F(v^k(t+h)) = q(v^k(t+h), u(t+h)) - q(v(t), u(t)) - 0.5h (g(v^k(t+h), u(t+h)) + g(v(t), u(t))) \quad (\text{EQ A.4})$$

and the Jacobian of $F(v^k(t+h))$, $J_F(v^k(t+h))$ is

$$J_F(v^k(t+h)) = \frac{\partial}{\partial v} q(v^k(t+h), u(t+h)) - 0.5h \frac{\partial}{\partial v} g(v^k(t+h), u(t+h)). \quad (\text{EQ A.5})$$

The only unknown in this system of linear algebraic equations is the vector $v^{k+1}(t+h)$, which is derived from $v^k(t+h)$ by solving the linear system of equations

$$J_F(v^k(t+h)) [v^{k+1}(t+h) - v^k(t+h)] = -F(v^k(t+h)) \quad (\text{EQ A.6})$$

using some form of Gaussian elimination or LU decomposition. The Newton iteration is continued until a sufficient degree of convergence is achieved, that is $\|v^{k+1}(t+h) - v^k(t+h)\| < \epsilon$ and $F(v^k(t+h))$ is close enough to zero.

A.3 Gaussian Elimination

One of the most efficient methods of solving a system of linear equations $Ax = b$ is by Gauss elimination and its derivative (EQ.A.6). The most common method in circuit simulation is LU decomposition. LU-decomposition involves the decomposition of the matrix A into two matrices, L a lower-triangular matrix and U an upper-triangular matrix, such that

$$A = LU \quad (\text{EQ A.7})$$

with A corresponding to the Jacobian matrix in equation (EQ A.6).

The new system

$$LU\Delta v = b \quad (\text{EQ A.8})$$

where Δv corresponds to $v^{k+1}(t+h) - v^k(t+h)$ and b is the right-hand side of the equation (EQ A.6), is solved in two stages.

First the system

$$Ly = b \quad (\text{EQ A.9})$$

is solved in the unknown y , using forward substitution. Then the system

$$U\Delta v = y \quad (\text{EQ A.10})$$

is solved for the unknown Δv by backward substitution. Finally the vector $v^{k+1}(t+h)$ is obtained as

$$v^{k+1}(t+h) = \Delta v + v^k(t+h) \quad (\text{EQ A.11})$$

Both Gaussian and LU decomposition require a number of multiplications which are of the order n^3 for the solution of dense matrices, where n is the dimension of the system of equations. However, most nodes in a circuit are connected to two or three other nodes, making the matrix very sparse. Therefore, for a large system, sparse matrix techniques are applied [33]. The advantages are that only the non-zero entries of the matrix are stored and that only the non-zero entries are processed in solving the matrix. Partial or full pivoting is performed to preserve the sparsity of the matrix.

A.4 The Truncation Error

Assuming that the numerical error introduced by the nonlinear iteration (EQ A.3) is negligible, the main source of error in the solution of $v^{k+1}(t+h)$ is due to the integration method, which provides a numerical approximation of the true solution.

This error is due to a combination of the roundoff error and the truncation error. The round-off error is the inevitable error that results from performing arithmetic operations on a digital computer, since finite precision arithmetic is used. The magnitude of the round-off error depends on the floating-point arithmetic unit of the computer being used. The truncation error results from the fact that the Taylor series is truncated after a number of terms and this error depends on the specific integration method. The estimation and control of Local Truncation Error (LTE), i.e. truncation error introduced in a single discretization step, is the key to the practical implementation of circuit simulation programs.

The LTE is used to control the time step very efficiently by optimally choosing a step size as large as possible provided the accuracy is achieved. The user provides two accuracy control parameters, ϵ_A and ϵ_R , which are the absolute and relative errors permissible in each integration step. They are combined to form a user error tolerance:

$$E_{userLTE} = \epsilon_A + \epsilon_R \max |x_{n+1}, x_n| \quad (\text{EQ A.12})$$

The BDF integration methods [Brayton72] uses an efficient strategy to compute the LTE involving the use of predictors. The LTE estimate is calculated between the computed solution x_{n+1} , and the predicted value $x^p(t_{n+1})$. For a k th-order BDF method, the following expression is used:

$$E_k = \left[\frac{h_n}{t_{n+1} - t_{n-k}} \right] (x_{n+1} - x^p(t_{n+1})) \quad (\text{EQ A.13})$$

Therefore, the computed solution x_{n+1} is accepted if $|E_k| < |E_{userLTE}|$.

The general form of the local truncation error for most multistep integration methods of order k is given by [Gear-71]

$$LTE_{n+1} = C_k h^{k+1} x^{(k+1)}(\xi) \quad t_n \leq \xi \leq t_{n+1} \quad (\text{EQ A.14})$$

where C_k is a constant which depends on the order of the integration method. Taking the ratio between the allowable LTE and the actual LTE

$$r = \frac{|E_{userLTE}|}{|E_k|} = \frac{|C_k h_{max}^{k+1} \ddot{x}(\xi)|}{|C_k h_n^{k+1} \ddot{x}(\xi)|} \quad (\text{EQ A.15})$$

assuming that both errors are $O(h^{k+1})$, it follows that

$$r = \left[\frac{h_{max}}{h_n} \right]^{k+1} \quad (\text{EQ A.16})$$

and

$$r_{LTE} = \frac{h_{max}}{h_n} = r^{\left(\frac{1}{k+1}\right)} \quad (\text{EQ A.17})$$

The r_{LTE} ratio is used to test for acceptance of the time step when $r_{LTE} > 1$ and the other to select the next step size

$$h_{n+1} = r_{LTE} h_n \quad (\text{EQ A.18})$$

Appendix B: Branch and Node Tearing in Circuit Simulation Decomposition

The following sections describe the Branch and Node tearing techniques for circuit decomposition found in third generation circuit simulators.

B.1 Branch Tearing

Branch tearing [6] is applied to a network by detaching suitable tie branches so as to create a number of unconnected subcircuits. To compensate for the branches removed by this process, currents equal to those that were flowing through the branches are injected at the nodes from which they were disconnected. This method is based on the nodal admittance approach. If we consider the network shown in figure 82, its corresponding admittance matrix may be written in the form:

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} \\ Y_{21} & Y_{22} & Y_{23} \\ Y_{31} & Y_{32} & Y_{33} \end{bmatrix} \quad (\text{Eq B 19})$$

The element of the off diagonal submatrices $Y_{ij}, i \neq j$ consists of the tie or transfer admittances connecting nodes of the subcircuits i and j . The nodal equations for the network may be written in the form

$$YV = I$$

(EQ B.20)

where V and I are vectors of node voltages and node currents, both of order n ,

FIGURE 82

Partitioned network and its torn representation

(a) Network partitioned into three subcircuits (b) The network after the branch tearing

respectively, where n is the total number of nodes in the circuit. By adding additional equations to the system, the tie admittances Y_{ij} can be removed from the Y matrix and the result is a BBD matrix of the form:

$$\begin{bmatrix} Y_{11} & 0 & A_1 \\ & Y_{22} & A_2 \\ 0 & & Y_{33} & A_3 \\ -A_1^T & -A_2^T & -A_3^T & Z \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ i \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ 0 \end{bmatrix} \equiv \begin{bmatrix} Y_s & A \\ -A^T & Z \end{bmatrix} \begin{bmatrix} V \\ i \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix} \quad (\text{EQ B.21})$$

where p is the number of branches removed, $Y_s(n \times n)$ is the block diagonal admittance matrix, $Z(p \times p)$ is the interconnection matrix, and $A(n \times p)$ represents the tie connections; therefore A has n rows, corresponding to each node of the original network, and p columns, corresponding to each of the tie branches. In column q of A , +1 will appear in the row corresponding to the node where $+I_q$ is injected and -1 in the row of the node at which $-I_q$ is injected as shown in figure 82b. In (EQ B.21) $i(p)$ is the vector of compensating currents to be injected, and V and I are the unknown node voltages and node currents respectively.

The solution of the BBD structure is similar for both tearing techniques. This will be explored in more detail in the node tearing section.

B.2 Node Tearing

This technique is similar to branch tearing, but instead of using branches for tearing the network, nodes are used for the decomposition. This approach is better because it is easier to tear and fewer variables are needed [60]. The nodes that are common between the subcircuits are the tearing nodes as shown in figure 83.

Each subnetwork can then be analyzed separately and the solution is obtained in terms of the tearing nodes. This is equivalent to a particular ordering of the nodal equations such that the resulting conductance matrix has a bordered block diagonal structure in which each block represent a subcircuit. The bottom diagonal block of figure 3a represents the interconnect nodes and the border represents the connection between the interconnect

nodes and the subcircuit. The way to solve this augmented matrix is equivalent to comput-

FIGURE 83

Network partitioning using node tearing

ing the conductance matrix that represents the Norton equivalent of the subcircuit at the tearing nodes. These matrices can be combined to form the interconnect matrix which is then solved to determine the voltages of the tearing nodes. The following steps are used to solve this BBD matrix in a nonlinear circuit:

1. Load all devices in each subcircuit to that subcircuit's matrix.
2. Solve each subcircuit matrix to obtain the conductance matrix that represents the Norton equivalent of the subcircuit at the interconnect nodes.
3. Add the Norton equivalents of the subcircuits together to obtain the interconnect matrix.
4. Solve the interconnect matrix to obtain the tearing node voltages.
5. Backward substitute each subcircuit to obtain the subcircuit variables.

Appendix C: A Slotted-Ring Bus Architecture Description and its Performance

C.1 Slotted-Ring Bus Description

Each PE incorporates an I/O processor that operates asynchronously to the main processor. The processor writes to other processors in a memory-mapped address space, and reads incoming data from other processors from a local dual-port memory (DPRAM). The transport of data via the ring bus is totally transparent to the processor. This concept is illustrated in Figure 84.

FIGURE 84

Unidirectional slotted ring bus

The term “slotted” arises from the presence of a clocked register in each PE's IOP. The IOPs of each PE are clocked by a master bus clock. This behaves as a circular shift register with parallel data packets. The slots are periods of the bus clock. An IOP may write a data packet into a time slot to be shifted down the bus to be read by other IOPs. The packet contains an address to allow other IOPs to determine if the data should be read from the bus and stored locally for access by the IOP of the processor element. Since the data on the bus has an associated address, it will be referred to as a packet. Thus, each packet consists of 32 bits of data and 13 bits of address for a total 45 bits. The mechanism for setting up a protocol to allow orderly passage of data is programmable at run time, allowing flexibility. The structure of the IOP is shown in Figure 85. The processor can transfer data to another processor by writing to the memory-mapped address block in the processor's local address space. The packet passes through a FIFO due to the asynchro-

FIGURE 85

Input/output processor description block

nous operating of the processor and bus, and is transported onto the bus by the IOP. This is transparent to the processor. The address of the packet is monitored by each IOP as the packet is shifted along the bus. If the address is for that PE or is a special broadcast address, then the IOP takes a copy of the data and places it into the dual port RAM at the address specified by the packet. Thus the packet address can be divided into two fields:

- PE address specifying a destination PE(s). The destination can be all PEs (i.e. a broadcast mechanism), and
- The DPRAM memory address within the destination PE.

The division between the two fields is programmable by the user.

This slotted bus overcomes the major disadvantage of the “standard” ring bus - latency. This is because there is no processor intervention in moving a data packet from its source to its destination - the IOP performs this transparently. The short interconnect path allows the bus clock to operate at low periods, which further reduces latency. For an N processor system the maximum bus transport latency is N-1 bus clock periods.

C.2 Bus Performance

The bus is designed to operate with a minimum clock period of 30 ns (33.3 MHz maximum). During this time period in the first time slot after the bus frame sync, each slot can be filled with a data packet from a different PE, and then shifted along the bus on subsequent periods of the bus clock. The bus is reloaded with new data upon the next frame sync and the shifting is repeated. Each data packet contains 32 bits of data (four bytes) and 15 bits of address. Due to the shifting, a bus communication latency is introduced. The actual bus bandwidth is dependant upon how the bus is configured in terms of its frame sync length. In the general case (as in the worst case) its frame sync and bus are set up to

allow communication between any PES. A best case peak data transfer rate can be obtained when the data flow is unidirectional to the adjacent PE only. Then the bus can be configured for a unity frame sync period, and new data can be transferred to the adjacent PE every bus slot (but not further than the adjacent PE).

In the general case, the bus frame length is the same as the number of PEs in the system i.e. there are N PEs and M bus time slots, where $N=M$. Each PE may write a data/address packet to the bus during the first time slot of a frame and then it is shifted $M-1$ times on the bus so it passes by each PE. During the first time slot, N packets are written, and then they are shifted for $N-1$ slots for a total of $M=N$ slots. Thus data is transferred at a rate of $N/M = N/N = 1$ packet per time slot. The the data transfer rate can be stated in a variety of ways, depending upon what type of system it is being compared to. For comparison to parallel data buses, the transfer rate is usually stated in data words or bytes per second. The transfer rate of this bus would be:

$$1 \text{ word}/30 \text{ ns} * 4 \text{ bytes/word} = 33.3 \text{ Mwords/S} = 133.3 \text{ Mbytes/S}.$$

When considering serial buses, the rate is usually stated in bits per second. This is the raw bit speed of the bus. For fair comparison between parallel and serial buses, the address bits should be included in the calculation when converting from a parallel bus speed to an equivalent serial bus speed. Accordingly, the transfer rate of the bus is:

$$(32 + 13) \text{ bits}/30\text{nS} = 45 \times 33.3 \text{ Mpackets/S} = 1.4985 \text{ Gigabits/S}.$$

The inclusion of the address bits in this calculation is fair because serial links must include address information pertaining to the destination of the packet of information.

Table C.1 shows the latency of communication between PEs for the minimum, average, and maximum cases. This table states the elapsed time from when a source pro-

cessor writes a data item to its bus output FIFO, until the time that data is received in the local DPRAM of the destination processor D logical positions away on the ring bus, in a N PE system.

TABLE C.1. Latency for Furthest Intercommunication Path in a N PE System

Delay for:	Minimum	Average	Maximum
Waiting for output slot	0	$N/2$	N
Time through bus output FIFO	1	1	1
Transport via bus	D	D	D
Time through bus input FIFO, write to DPRAM	2	2	2
Total (Bus cycles)	$D + 3$	$N/2 + D + 3$	$N + D + 3$

In Table C.1, D is the distance along the bus of the destination PE (in the direction of the bus shifting) and N is the number of PEs installed in the system. The minimum distance is to the next PE in the direction of the bus data flow. The maximum distance is to the adjacent PE in the direction opposite the bus data flow, and the average is for communication to the PE $N/2$ positions away. With $N=4$, the average communication for a single word transfer between two processors which are not adjacent requires 7 bus cycles. With a 30 ns bus cycle, this average transfer is 210 ns.

END

1 4 1 1 9 5

FIN