APPLICATIONS OF PARALLEL PROCESSORS TO TECHNOLOGY COMPUTER-AIDED DESIGN PROBLEMS

by

Eric Ramos Tomacruz

Memorandum No. UCB/ERL M94/101

14 December 1994

COVER PAGE

APPLICATIONS OF PARALLEL PROCESSORS TO TECHNOLOGY COMPUTER-AIDED DESIGN PROBLEMS

by

Eric Ramos Tomacruz

Memorandum No. UCB/ERL M94/101

14 December 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

APPLICATIONS OF PARALLEL PROCESSORS TO TECHNOLOGY COMPUTER-AIDED DESIGN PROBLEMS

by

Eric Ramos Tomacruz

Memorandum No. UCB/ERL M94/101

14 December 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Applications of Parallel Processors to Technology Computer-Aided Design Problems

by

Eric R. Tomacruz

Doctor of Philosophy in Engineering

University of California at Berkeley

Professor Alberto L. Sangiovanni-Vincentelli, Chair

The feature size shrinkage of integrated circuits has made accurate three dimensional modeling and simulation of semiconductor processes and devices indispensable. The simulation problem consists of finding the solution to a system of PDEs. This dissertation uses the control volume approach which discretizes the simulation region into a set of subvolumes each represented by a grid point. This step transforms the PDEs into a nonlinear set of equations. Numerical integration methods are then applied to the time dependent derivatives and each equation is linearized through the use of the Newton-Raphson method. Finally, each system of linear equations is solved using the iterative Conjugate Gradient Squared (CGS) method. Since the number of equations is proportional to the number of grid points, three dimensional simulation is computationally intensive. This dissertation investigates the applicability of parallel processors to carry out 3-D process and device simulations. Since most of the total CPU time is spent on the linear system solution, a major effort is placed on developing and implementing parallel preconditioned linear solvers. The solution process efficiency is improved through the use of multigrid methods, irregular grids, and adaptive grids. Silicon pixel detectors and chemically amplified resists are studied in detail. The parallel processors used for this study are the single instruction multiple data CM-2 and the multiple instruction multiple data CM-5.

[illegible]

Professor Alberto L. Sangiovanni-Vincentelli

Committee Chairman

Acknowledgments

I would like to thank my research advisor, Professor Alberto Sangiovanni-Vincentelli, for his guidance on my research for the past five years. His critical evaluations of my ideas and my papers over the course of my graduate career have truly improved my research abilities. It has been a privilege to work with him. I would also like to thank Professor Andy Neureuther for his support and guidance in my resist simulation research. My work in the area has widen my perspective in technology CAD problems. I would also like to acknowledge Professor Alexandre Chorin and Professor Phil Colella for their ideas in solving the drift-diffusion equations using explicit methods. Thank you to Professor Parlett and Professor Demmel for their help in analyzing the linear solvers used in this dissertation.

Most of the ideas and results presented in chapter 2 would have not been possible without the help of Professor Roberto Guerrieri, Toru Toyabe, Don Webber, and Francesco Forti. Professor Guerrieri is responsible for several key ideas in implementing the CM-2 device simulator and the multigrid extension. The mismatch of Toru Toyabe's sequential code and my parallel code resulted in the development of the partitioned natural ordering preconditioner. Don Webber and I implemented the first and probably the only Lisp drift-diffusion device simulator in existence. Francesco Forti provided all the experimental results for the silicon pixel detector. I also wish to thank Alan Torres, Eugene Loh, and Terry Dontje of Thinking Machines Corporation for the use of the Los Alamos Connection Machines.

The research presented in chapter 3 was done in cooperation with Jagesh Sanghavi. His superior organizing and programming abilities accelerated the research significantly. Thank you to Horst Simon for providing the spectral partitioner, Anna Pierantoni for providing hydrodynamic matrices, Gernot Heiser for answering my Second questions, Claude Pommerell for helping me install PILS, Nancy Hitschfeld for answering my Omega questions, and Alper Demir for helping me generate circuit noise simulation results. I also acknowledge Eric Fraser and Mark Kremenetsky for all their help in using the CM-5.

The work presented in chapter 4 was done with Marco Zuniga and David Newmark. Marco Zuniga provided most of the experimental results essential for the verification of the diffusion models. David Newmark did the proximity effects experiment which validated the usefulness of the chemically amplified resist simulator. I would also like to thank John Hutchinson for supplying me the PTBOCST and PTBMA models.

Thanks to all the members of the CAD group. In particular, I would like to thank Henry Sheng for constantly helping me throughout the years. I also acknowledge the help of Hitoshi Matsuo, Stephen Edwards, Elise Mills, and Flora Oviedo.

Finally, I would like to thank my family and Bambi Santos for all their support and understanding. Life as a graduate student is very demanding. Thank you to Dad, Mom, and Bambi for all the confidence and patience they have given me.

This research is funded in part by the Defense Advanced Research Projects Agency under contract N00039-87-C-0812/0817, NSF infrastructure grant number CDA-8722788, Thinking Machines Corporation, Hitachi, SGS-Thomson, and the California state MICRO program.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction to Technology CAD and Parallel Processors

1.1 Technology Computer Aided Design

1.1.1 Role of Particle Distribution

Technology computer-aided design (TCAD) is essential to the development and fabrication of advanced integrated circuits. It reduces development time and cost by allowing engineers to explore different design and process possibilities without the need for actual implementation. It also allows the design of hypothetical structures and processes that are not possible with the current technology. TCAD is divided into two main areas - process and device simulation. Figure 1.1 illustrates the physical characteristics of the device structure as the end product of process simulators and the electrical characteristics of the device structure as the output of device simulators. The electrical characteristics may then be used as an input to circuit simulators.

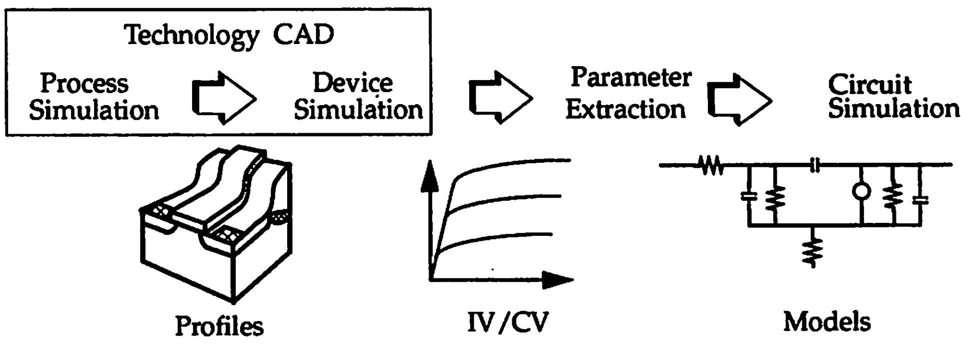

Figure 1.1: Technology Computer-Aided Design

A main goal of IC process and device simulation is to find the spatial distribution of mobile, charged or neutral particles as a function of time. The particles are electrons and holes for device simulation and are atoms and molecules for process simulation. The distribution of mobile particles is governed mainly by particle transport equations and Maxwell's equations. There are three common methods to solve these partial differential equations (PDEs) - particle method, control volume, and finite element. The focus of this dissertation is in utilizing the control volume approach which divides the simulation space into subvolumes with aggregate variables that describe the average state of particles in the subvolume.

Control volume approaches in TCAD are usually used to solve the continuity equation.

$$\frac{\partial C}{\partial t} = -\nabla \bullet F \tag{1.1}$$

where C is the concentration of particles and F is particle flux density. Equation 1.1 states that the time rate of change of the total amount of a particle contained in a control volume is equal to the inward flux of the particle across the boundaries of the control volume. The particle flux density may be described by

$$F = D\nabla C + \frac{q}{kT}DC'E \tag{1.2}$$

where D, C', and E are the diffusivity, electrically active particles, and electric field respectively. q, k, and T are the electric charge constant, Boltzmann's constant, and temperature, respectively.

Process simulation may be grouped into four categories - lithography, etching, thermal processing, and deposition. The continuity equation is solved in the area of thermal processing specifically in the area of oxidation, dopant diffusion, and acid diffusion. One difficulty in solving this equation lies in the specification of D. For impurity diffusion, D may be concentration-dependent in order to model diffusion mechanisms that are controlled by physical quantities such as vacancies and excess interstitial concentration. For acid diffusion, the diffusivity may also be concentration-dependent due to interactions between acid and activated sites. This will be discussed further in Chapter 4.

Device simulation involves the solution of the Boltzmann Transport Equation (BTE) which can be obtained by generalizing the continuity equation [2]. For control volume methods, there are two simplifications of the BTE that are commonly solved - drift-diffusion equations and hydrodynamic equations. The drift-diffusion equations consist of the carrier continuity equations and Poisson's equation. The hydrodynamic model extends the drift-diffusion model by generalizing some variables in Equation 1.2 and adding a third term to represent a contribution from drift energy. The device simulation equations are further discussed in Chapters 2 and 3.

1.1.2 Current Computationally Expensive Simulation Problems

State of the art TCAD simulations involving the control-volume method are very computationally expensive. For example, a CMOS latch-up simulation takes 5 hours on a Cray-2 [3]. Nishi and Ueda [4] report the use of deep-submicron CMOS circuits to investigate the effects of narrower spacings to propagation delay and also to get an optimal process conditions with respect to gate oxide thickness and threshold voltage. CPU time for the simulation of one sample on a 30 MIPS machine is 2 hours for process simulation and 2.5 hours for device simulation. Leakage currents due to parasitic MOS effects were studied by Noell et al. [5]. Using a 48,190 mesh size, a 3-D simulation to generate a single current-voltage curve took 8 hours on a Multiflow Trace 14/300 [30]. Trench-bounded MOSFETs found in DRAM cells having a deep trench storage capacitor were simulated by Knepper et al. [6]. Mesh structures with up to 100,000 nodes may be required to simulate accurately submicron features of the structure. Run times on the order of 15-30 minutes per bias point are required by an IBM RS/6000.

Although traditional vector machines may provide the computational power needed for TCAD simulations, parallel machines may provide a good alternative. The effective use of parallel machines for TCAD involves a good understanding of parallel machines, parallel programming methodologies, and the target TCAD application. The remaining sections of this chapter provides an overview of parallel processors and applications that have been developed for these machines.

1.2 Parallel Processors

1.2.1 Evolution

Due to progress in microelectronics, high-density packaging, advanced processors, memory systems, and other hardware technology, parallel processing architectures with hundreds or thousands of processing elements are now possible. The evolution can be traced back four decades to the introduction of the first sequential architecture. This architecture gradually improved through the use of lookahead functions which allowed the overlapping of instruction fetch, decode, and execution. The lookahead function eventually matured to the more general pipelined architecture. Then came the advent of vector processing which performed arithmetic or logical operations on scalar data items. The independent computations allow very deep pipelines and minimize instruction fetch. Also, vector processing allowed faster memory access and control hazards from branches are made nonexistent.

Parallel machines have been around since the 1960 (Burroughs D825 - 4 processors). Massively Parallel Processor (MPP) systems started in 1968 with the release of the Illiac IV computer, which had 64 processing elements (PEs) under one controller. However, in the early stages the machines did not have adequate technology for the architectural concepts involved. To attract users, a parallel processor must offer substantially better performance over computers existing at the time. A new technology must not only catch, but also surpass an entrenched technology in order to replace it. This is very difficult while the entrenched technology is still making good progress.

Today, the progress of the high end von Neumann machine has slowed down. In terms of clock rate, as shown by Figure 1.4, the current fastest von Neumann machine is only a few times faster than the current state of the art microprocessor. It is not clear if technological barriers for high end von Neumann machines are more difficult compared to microprocessor development technological challenges. However, there is definitely more market demand for microprocessor-based computing platforms. More financial resources are available for microprocessor development due to economies of scale. Hence, designers of high end machines are attempting to remove traditional barriers and gain power through scalable use of hundreds or thousands of off-the-shelf processors.

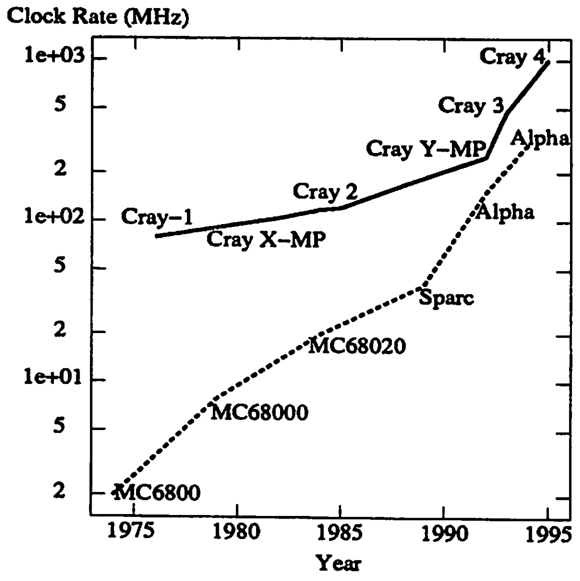

Figure 1.2: Clock Rates for Supercomputers and Workstations

1.2.2 Hardware Model

1.2.2.1 Flynn's Classification and Beyond

Flynn [7] introduced a classification of computers in 1972. He divided architectures into four categories - SISD, SIMD, MIMD, and MISD. S, I, M, and D denote single, instruction, multiple, and data respectively. Thus, SIMD means single instructions multiple data. SISD such as personal computers and most workstations are the most common. Vector processors and some parallel processors such as the CM-2 and the MP-1 fall under SIMD. SIMD architectures only require one program. The main distinction between parallel and vector SIMD is that more parallelism is available with parallel SIMD - tens vs. hundreds or thousands of simultaneous operations. MIMD architectures can be viewed as SISD architectures with a special communication network between processors. This architecture may have many programs and hardware construction can be simplified using off-the-shelf uniprocessors. Also, with this architecture, communication can be merged with computations. MISD machines are not common.

A different program for each MIMD processor is difficult to implement. Hence, a new term "SPMD" which stands for single program multiple data has been introduced. This classification denotes that each processor works on its own data and each processor may be in different parts of the program at one specific time. There are two variations of SPMD. One variation simplifies programming with coordinated and separate communication. The other has merged communication which allows the overlapping of computation and communication. SPMD with separated computation and communication implies programming that is as simple as SIMD since communication is always synchronized. SPMD with merged communication require less programming effort than general MIMD.

The naming convention for dedicated parallel computing platforms has evolved over the past three decades. Originally, they were referred to as parallel computers or parallel processors. With the introduction of the CM-2, the term massively parallel processors (MPP) became the standard naming convention. Today, if the number of processors is significantly lower than SIMD architectures, the machines are just referred to as parallel processors (PP) or sometimes multiprocessors (MP).

1.2.2.2 Memory

Parallel machines can also be classified according to the way they access memory - shared memory and distributed memory. The Cray Y-MP is an example of machine with shared memory. Each of its processors may access the same memory location. For larger parallelism, distributed memory which means each processor has its own local memory is common. There are several advantages to using distributed memory.

First, distributed memory machines are easy to build. Second, many applications have locality. Third, using nearby memory is five to fifty times faster than a centralized memory. And fourth, current PP hardware and software are local memory oriented. Hence, distributed memory is a key to portable code. Distributed memory has a disadvantage since compilers need to know how to use nearby memory. This may be difficult with traditional sequential programs. Hence, programmers need to learn to use nearby memory for speed.

Another way to classify parallel machines is through their address space - local and global. Intuitively, distributed machines should have local address space. Some distributed memory machines such as the Cray T3D and the KSR-1 have special hardware that connects local memories directly. This allows a global address space in which a processor can access another processors memory without message passing routines between processors. With special compilers, machines with local memory can also be programmed with a global address space.

1.2.2.3 Interconnection Network

The interconnection between processors is another aspect that differentiates one parallel machine from another. There are several criteria for a good interconnection network. The relative importance of each criteria is determined by the target application and the efficiency of the tools for algorithm implementation.

1. Functionality - the machine's ability to support efficient data routing, interrupt handling, synchronization, and combine functions.
2. Network Latency - the minimum time delay to transfer data through the network.
3. Diameter - the shortest path between any two nodes.
4. Bisection Bandwidth - the minimum number of cut edges for a given a network cut equally into two.
5. Hardware Complexity - translates to implementation cost.
6. Scalability - the expandability of the network.

Figure 1.3 illustrates four example communication networks currently used today. Table 1.1 illustrates the characteristics of these networks based on the criteria described earlier. Several observations can be made in interpreting this table.

1. Most networks have nodes with small degree.
2. Hypercube node degree increases with $\log_2 N$ which is bad with large N

3. High degree of concurrent communication may be more important than the diameter.
4. The number of links and node degree affect network cost.
5. Bisection width can be enhanced by a wider channel width.

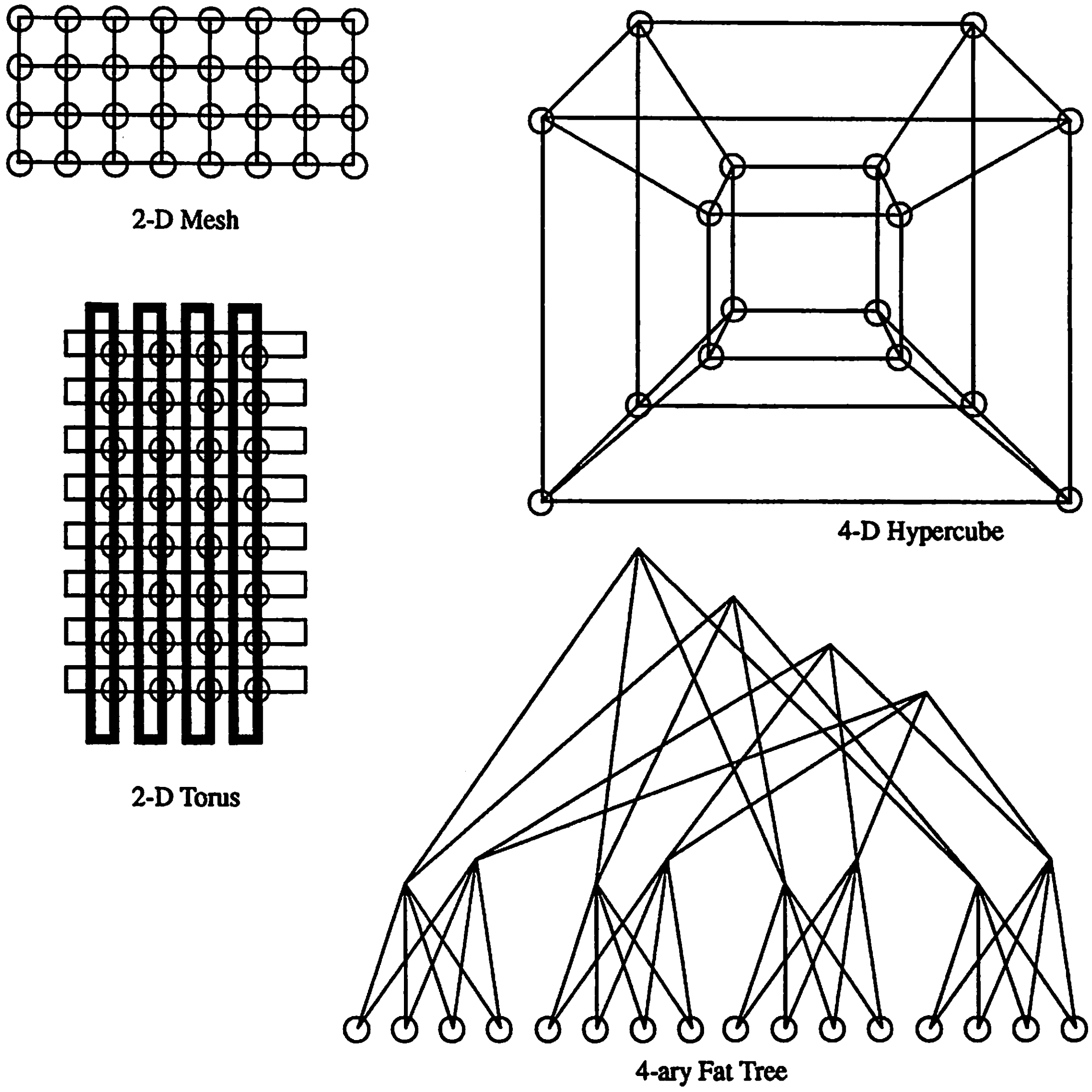

Figure 1.3: Interconnection Network Examples

Table 1.1: Network Characteristics

Network Type	Node Degree	Diameter	Number of Links	Bisection Width	Remarks on Network Size	Sample Machines
2-D Mesh	4	2(r-1)	2N-2r	r	r = sqrt(N)	Paragon
Hypercube	n	n	nN/2	N/2	n = $\log_2 N$ (dim)	iPSC/2, nCUBE, CM-2
Fat Tree	2	2h	$4N(1-2^{-h})$	$2^{(h+1)}$	h = $\log_4 N$	64 Node CM-5
3-D Torus	6	r/2	3N	$2r^2$	$r^3 = N$	T3D

1.2.2.4 Current Trends and Target Architectures

A totally coherent taxonomy of parallel processors is difficult to construct. Each aspect of architectural consideration influences each other to some degree. Architectural research and development goals include packaging efficiency, scalability, adaptability to current technology, and backward compatibility. Current parallel architectures appear to be converging to a generic parallel architecture. The generic parallel architecture uses off-the-shelf processing technology, uses distributed memory, and has a simplified interconnection network. By using off-the-shelf processing technology, development cost is reduced. Survival of interconnection network in future systems depends on packaging efficiency and scalability. Hence, parallel processor companies such as Intel and Thinking Machines Corporation have switched from the complicated hypercube to much simpler interconnection networks. Cray and I.B.M. which only recently entered the parallel processing market also have introduced machines with simple communication networks.

The Connection Machine 2 (CM-2) and the Connection Machine 5 (CM-5) MPPs are used to implement algorithms presented in this dissertation. The CM-2 is representative of the SIMD technology while the CM-5 has a MIMD architecture.

The Connection Machine 2 is a massively parallel computer with up to 65536 processors with a conventional computer as a front end. Each processor is bit serial which have a clock rate between 7 to 10 MHz and can have 64k to 1024k bits of local memory. The processors may be equipped with floating point accelerators which are shared by a cluster of 32 processors. The CM-2 is a Single Instruction Multiple Data (SIMD) architecture. The nodes exchange data among themselves through the router, NEWS grids, or a scanning mechanism. Each processor chip contains 16 processors and a router. The router nodes are wired together to form a Boolean n-cube. The NEWS grid is based on the fact that each processor has a north, east, west, and south neighbor in the various grid configurations. The CM-2 also has special hardware for fast data combining or spreading throughout the entire array [16].

The Connection Machine 5 is a massively parallel computer in which data parallelism can be implemented in either a SIMD mode, or synchronized Multiple Instruction Multiple Data (MIMD) mode. Each node of the CM-5 is a SPARC microprocessor with up to four vector units and 32 Mbytes of memory [17]. The Sparc processor has a clock of 33 MHz and a 64 Kbyte cache. The processors are interconnected using three networks: data network, control network, and diagnostic network. The data network is configured as a 4-ary fat tree. Each processor has two connections to the data network which correspond to a bandwidth of 40 Mbytes/s in and out of each leaf node. An aggregate bandwidth of 160 Mbytes/s out of a subtree is achieved with 16 leaf nodes since only two parent connections are needed. For four parent connections, a bandwidth of 10 Gbytes/s is obtained. The bandwidth continues to scale linearly up to 16,384 nodes [16].

1.2.3 Programming Model

There are three ways for a user to program a parallel processor. The first is through the use of parallelizing compilers that automatically extract parallelism from programs written in existing serial languages. The programmer may aid this process by specifying data partitioning and assisting code optimization. Second is through the use of a serial language augmented with a new constructs that allow the programmer to specify and properly coordinate the execution of parallel tasks. Finally, the programmer could write in an entirely new language designed to make parallelism easier to detect and extract. This can be done by eliminating side effects and other features that make it difficult to extract parallelism from programs written in serial languages. Implicit parallelism involves functional programming, dataflow, Prolog, and other "side-effect-free" languages.

Significant research is also being done in the area of parallel tools and languages. Ease of programming and debugging in obtaining high performance portable code is the main goal. Pancake [33] surveyed Supercomputing Conference Proceedings and observed that current programming methodology focuses on the data parallel and message-passing approaches which are used to implement algorithms in this dissertation. These approaches are based on programming languages extended to handle the execution of a parallel task.

In the data parallel computing model, each processor has some memory associated with it. The processors may act under the direction of a serial computer called the front end. Each processor stores the information for one data point in its local memory; all processors can then perform the same operation on all the data points at the same time. A programmer can specify that only a particular subset of the processors is to carry out an operation. Data parallel algorithms map well to SIMD architectures such as the CM-2.

In the message-passing environment, each processing node runs an independent copy of single program, and manages its own computations and data layout. Communication between nodes is handled by calls to communication library routines. These routines allows a user to send messages from one processing node to another in a number of different ways. It also provides function for point-to-point messaging and global operation. It may also provide low-level tools for "active message" operations which allow the overlapping of computation and communication. One of the most useful features of message-passing programs is that they allow the processing nodes to synchronize as frequently or infrequently as required for a given application. Message-passing algorithms map well to MIMD architectures such as the CM-5.

PVM (Parallel Virtual Machine) is a software package for using a heterogenous network of computers as a single computational resource [34]. Due to current large latencies in representative workstations and LANS, PVM is designed to provide a message-passing environment for applications with relatively loosely coupled, large grain parallelism. However, for homogenous network of processors such as the CM-5, it also provides a good message-passing environment for highly coupled parallel applications [35].

1.2.4 Parallel Processing Performance

1.2.4.1 Peak Performance

Computer manufacturers usually state peak performance in terms of MIPS (millions of instructions per second) or Mflops (millions of floating point operations per second). Figure 1.4 illustrates the peak performances of vector machines and parallel machines in terms of Mflops [30]. The trend shows an order of magnitude increase in performance for parallel machines compared to vector machines. Although vector supercomputers like NEC's SX-3 (4 processors) and Hitachi's S-3800 (4 processors) whose claimed peak of 20 gigaflops and 32 gigaflops respectively may offer the computational power needed, massively parallel processors (MPPs) that offer a peak of over 100 gigaflops provide an attractive alternative. The gap between MPP and vector machine peak performances is expected to widen. Hence, as shown by Table 1.2, manufacturers of traditional vector machines are currently shifting their product line to parallel vector machines. Supercomputers in the future will unlikely have less than 16 processors. As a result, the computational requirement of current and future sophisticated semiconductor device simulation problems will be satisfied by parallel machines.

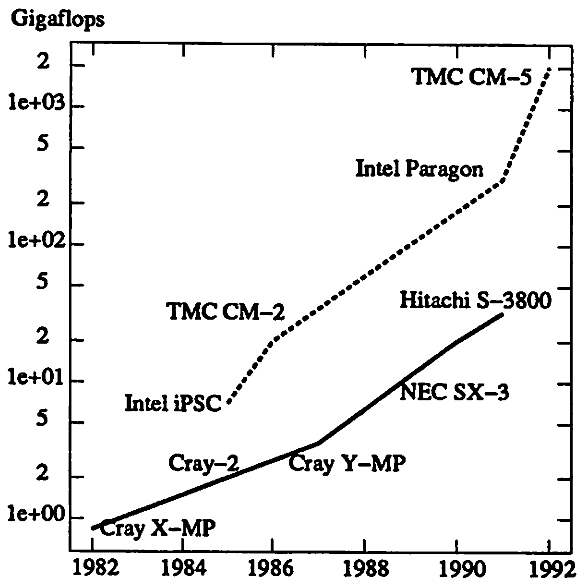

Figure 1.4: Peak Computation Performance

Table 1.2: Current Parallel Vector Machines

Parallel Vector Machine	Year of Introduction	No. of Processors	Peak GFlops
NEC SX-4	1994	16-512	1024
Cray-4	1995	16-128	256
Hitachi SR2001	1994	8-128	?

1.2.4.2 Sustained Performance

As mentioned earlier, computer manufacturers usually state peak performance in terms of MIPS or Mflops. These performance metrics are by no means conclusive. The real performance which can also be called sustained performance is application-driven and program dependent. Dongarra [38] has shown the effectiveness of parallel processors for the Linpack benchmarks. This benchmark may identify key properties such as floating point performance but is not an accurate predictor of general purpose performance. The infancy of compiler technology for parallel machine is another issue to consider. Sethian [39] reports a factor of five speed-up by just recompiling applications a year later.

To make statements on the usefulness of parallel machines for real applications such as TCAD, it is necessary to actually design and implement the algorithms. It should be pointed out the best algorithms for sequential machines are not necessarily the best algorithms for parallel computers. The best measure of performance is the wall clock time which is the time it takes a machine to solve the problem. For parallel machines, the algorithm with the best computational efficiency may not necessarily give the algorithm with the least wall-clock time.

1.2.5 Clusters of Workstations Versus Multiprocessors

Due to its general use, workstation clusters are widely available and can be viewed as MPPs with a slower network. Their cost is lower because of economies of scale leveraged across the entire workstation user community and because of their dual use - normal workstation and parallel processor. Also, as pictured in Table 1.3 [40], MPPs are usually more than a year behind in terms of utilizing the state of the art microprocessor. Since microprocessors improve 50% per year in terms of speed, a one year lag would result in a 1.5 factor degradation of an MPP node compared to a workstation. Hence, workstation clusters are very attractive for parallel processing. However, the applicability of clusters of workstations will be determined by global system software, communication network, and the target application.

A global system software treats a collection of processors, memory, and disks as a single machine. To reduce development time, the use of off-the-shelf technology such as existing operating systems is a good approach. One can just add communications protocol software and a global system layer such as PVM and one can already implement parallel software. However, as will be shown later, existing software layers degrade the performance of communication networks.

Table 1.3: MPP Technological Delay

MPP	Processor	Year	Workstation
T3D	150 MHz Alpha	93/94	92/93
Paragon	50 MHz i860	92/93	91
CM-5	32 MHz SS-2	91/92	89/90

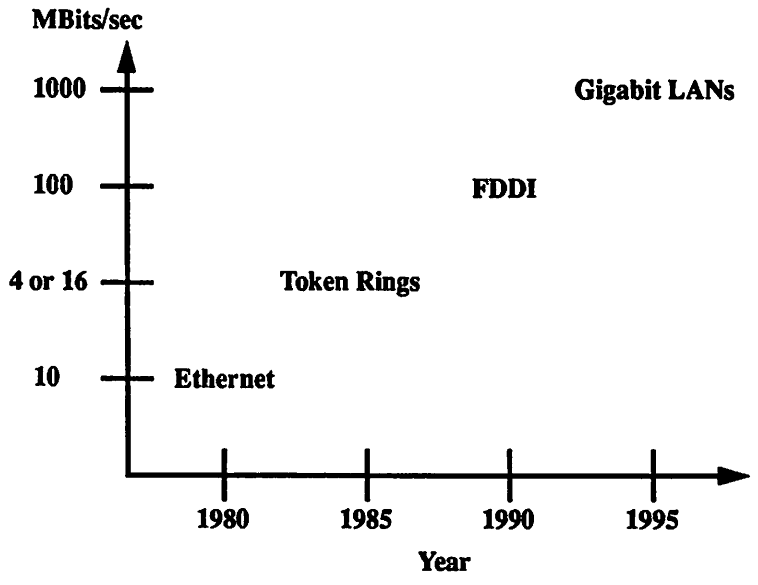

Figure 1.5: Evolution of LAN Data Rates

Table 1.4: Data Rates and Latency for Some MPPs

MPP	Year Introduced	Data Rate (Mbits/sec)	Latency (ms)
Intel iPSC	1985	10	1.7
Intel Paragon	1991	1600	0.08
TMC CM-5	1992	320	0.01
Cray T3D	1994	2400	?

For workstation clusters to execute efficiently fine grain parallel algorithms, it must address two key communication issues. First, its communication network should have a high data rate transfer. Figure 1.5 shows peak data rates for local area networks (LAN) [31] and Table 1.4 illustrates sample data rates for MPPs. LAN data rates are comparable to MPP data rates. Second, LAN networks should have low-latency. A physical limit for latency is the speed of light [32]. Information travelling at the speed of light takes 70 microseconds to go halfway around the globe. In terms of hardware considerations, no published theoretical limits are available since assumptions have to be made about the network topology, hardware implementation, and the manner in which messages are sent.

Tables 1.5 and 1.6 present the actual communication and global operation speeds of workstation clusters using PVM [35]. Two important points can be observed. First, the latency is in the order of milliseconds which is two orders of magnitude slower compared to MPPs. Second, actual bandwidth could be driven at near theoretical peak capacity for large messages.

The slow performance presented in Table 1.5 for Ethernet and FDDI networks is due assumptions made in traditional LAN software. These include invoking the operating system on every message, driver support for complex gather/scatter operations, and protocols which model communication only in point-to-point terms rather than in an all-to-all framework. The costs of all operations of the communication software including context switching buffer and timer management, scheduling, and data copying must be reduced to improve performance.

Latency can be hidden by using active messages [37] such that while one process is waiting for a response, another process which does not depend upon this response, may proceed with its processing. Martin [36] and von Eicken et al. [37] present active message implementations on workstation clusters with special communication hardware.

Table 1.5: Data Transfer Times (milliseconds)

Network	Message Length					
Type	0	128	1K	16K	64K	1M
Ethernet	1.2	1.5	3.2	24.5	82.3	1211.2
FDDI	1.2	1.5	2.5	16.1	60.3	665.7

Table 1.6: Global Operation Times (milliseconds)

Operation	No. of Networked Processors		
Type	2	8	32
Barrier	2.2	28.1	107.2
Broadcast	3.2	15.9	65.9
Opt. Bcast	1.2	11.5	35.1

Martin [36] presents the HP active message layer (HPAM) which is a software layer that delivers close to the hardware performance to user level programs. The difference between HPAM and typical LAN communication software are the following.

1. Direct user access to the hardware
2. An all-to-all, request-reply model of communication

3. Careful management of the network state to keep the overhead low.

Martin [36] describes active message measurements on a network of 4 HP 9000/735 workstations with Medusa FDDI interface cards. HPAM achieves a round trip time for a 20 byte payload of 29 *usec* and a maximum bandwidth of 12 MB/s. The main limitation to scaling, independent of the network, is the buffer requirements. It requires $4*D*P$ buffers per node where D and P are the network depth (number of outstanding packets to get full bandwidth) and number of processors respectively. For a shallow network, $D = 2$ or 4, it would scale well to 64 processors. However, it would not scale to a network with 1000 processors.

von Eicken et al. [37] evaluates a prototype implementation of the low-latency active messages communication model on a Sun workstation cluster interconnected by Fore Systems SBA-100 ATM interfaces using a 140 Mb/s TAXI fiber. Measurements show application-to-application latencies of about 29 microseconds for small messages which is comparable to the CM-5. The speed comes from a careful integration of all layers, from the language level to the kernel traps. The key issues are avoiding copies by having the application place the data where the kernel picks it up to move it into the device and by passing only easy to check information. Again, the cost is the large pre-allocated and pinned buffers which does not scale well to a large number of processors.

Future networks will definitely improve but it is not clear if Martin [36], von Eicken et al. [37], or other network implementations will be used. As network latencies improve, more multiprocessor applications will be ported to workstation clusters. Section 3.5.1 discusses the issues and possible solutions for a workstation cluster implementation of a TCAD application.

1.3 Applications of Parallel Processors

1.3.1 General Applications

Numerical processing and symbolic manipulation are two broad categories of applications that may be done in parallel. Scientific and engineering applications offer some very large numeric computation problems which includes particle calculations (plasmas), fluid dynamics (weather, aircraft design), and computer-aided design. Symbolic processing applications include database systems and applied artificial intelligence. Figure 1.6 illustrates the grand challenges identified in the U.S. High-Performance Computing and Communication (HPCC) program. This diagram shows the levels of processing speed and memory size required to do both numerical and symbolic manipulations. A significant number of these applications have been successfully solved in parallel machines. A brief survey of these problems with some similarity to TCAD problems will be discussed.

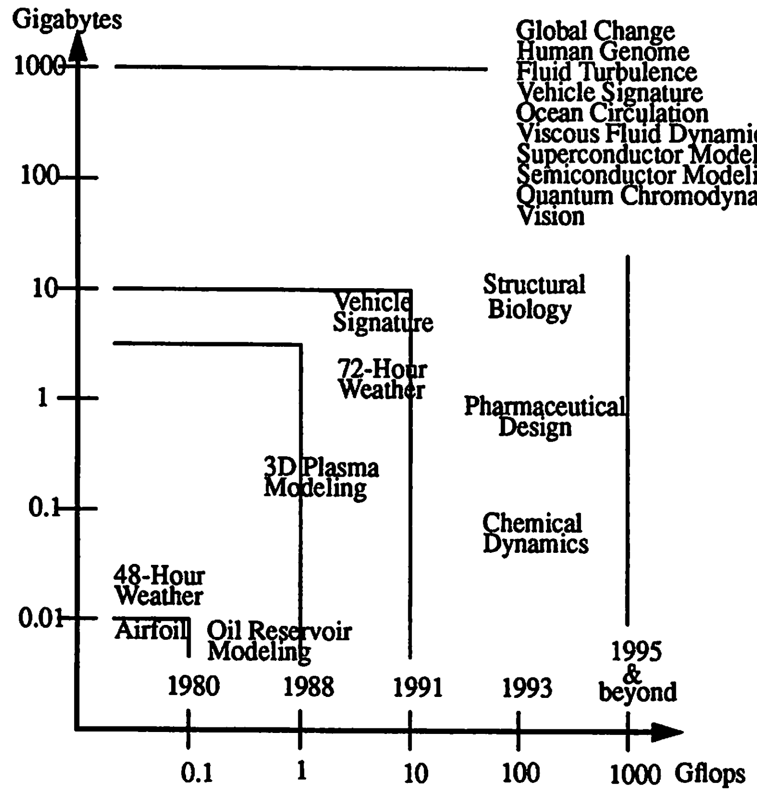

Figure 1.6: Grand Challenge Computing Requirements (U.S. High Performance Computing and Communication Program, 1992)

Considerable effort has been expended in weather simulations due to their benefits to the economy and the quality of life. Numerical weather predictions are defined as follows: Given the current state of the atmosphere (i.e., the values of certain specified meteorological quantities), calculate the state at various time in the future. The state of the atmosphere is defined by eight quantities - two horizontal components of wind velocity, temperature, humidity, shifted surface pressure, geopotential, vertical wind velocity, and pressure. The model consists of the relevant laws of nature expressed as partial differential equations - horizontal momentum equations, continuity equation, equation of state, first law of thermodynamics, humidity equation, and the hydrostatic equation. Explicit time integration methods are used to solve the equations that are discretized using rectangular grids. Parallel weather code implementations have been described by Korn et al. [41] and Dennis et al. [42].

Of great importance to the petroleum industry is the acquisition of undistorted subsurface image from seismic echo data using seismic migration. By starting with a rough velocity profile, the seismologist uses a cut-and-try iterative procedure to generate and output image that is consistent with measurements. A major difficulty in seismic migration as compared to other image processing techniques is that propagation velocity of the seismic signals can vary by a factor of ten from one part of the seismic image to the other. Typical solution methods use finite difference techniques to solve the wave equation and to extrapolate the signals recorded at the surface downward into the desired region; that is, it solves the wave equation to find out what subsurface features gave rise to the echo pattern recorded at the surface. Fourier transforms are used to solve the wave equation by transforming the problem from the time domain to the frequency domain. This causes the frequency to be the outer loop variable which may be evaluated in any order. Hence, the iterations in the outer loop becomes a pool of tasks that can be performed in parallel [43] [44].

The time evolution of a system of *n* bodies, each interacting with all other bodies by gravitational attraction or some other symmetrical force, is another computationally intensive problem. Typical solution methods [45] for message-passing programs given *n* bodies can be described as follows: Let each body be mapped to a unique processor. The innermost loop computes the gravitational force between the host body and $(n-1)/2$ visiting guest bodies. The other code in the inner loop shuttles the guest bodies in and out. The next outer loop sends the host body's clone out to visit other processes, combines its baggage with that of the half-updates clone that stayed at home. With this method, each processor only needs to send and receive a total of four messages. For problems with more bodies than processors, the same communication pattern can be used. However, communication and load balancing issues need to be addressed.

Quantum chromodynamics (QCD) is a theory about particles that make up atomic nuclei. QCD provides a formula for the probability that any specified configuration of quarks and field at one instant will arrive at another specified configuration at some later instant. The space-time continuum is approximated by an $N \times N \times N \times N$ four dimensional rectangular lattice, and the problem is reduced to the evaluation of an integral of over $56 \times N^4$ variables [46]. Deterministic methods of numerical integration such as the trapezoidal rule would require astronomical amounts of time even on small lattice structures. Hence, integration methods based on Monte Carlo statistical sampling are used. Butler et al. [47] presents a Monte Carlo procedure similar to a parallel Monte Carlo semiconductor device simulator which is described in Section 1.3.2.

Catlett [48] presents the use of gigabit networks to treat multiple computing resources as single system rather than a network of computers. The applications are organized into three general types - computational science, data navigation, and collaborative environments. In computational science, successful applications such as a coupled atmosphere-ocean general circulation model have been implemented on supercomputers that are linked together. The use of scientific workstations and the Parallel Virtual Machine

(PVM) have been attempted on tightly coupled algorithms. The efficiency of the algorithms depends mainly on the amount of computation between communication calls.

Majumdar and Martin [49] present a parallel preconditioned conjugate gradient algorithm applied to a neutron diffusion problem. The paper describes an implementation on a distributed workstation (IBM RS6000) environment using the PVM parallelization software. They claim a very good result of 70% efficiency for a one-dimensional fixed-source neutron diffusion problem on a cluster of 7 workstations. However, there are several points not discussed about the implementation. First, a single processor takes 4927s to solve a 701 node problem. This is an order of magnitude longer compared to the solution of device matrices. Second, they did not explain the implications of doing 2-D and 3-D simulations. Such simulations may require a lot more communication calls. Third, there is a linear drop in efficiency from one processor to seven processors. If the trend continues, efficiency for large clusters of workstations would be very poor.

Some algorithmic aspects developed for TCAD may be useful to other fields and vice versa. Generic routines such as parallel PDE solvers and Monte Carlo integrators are useful in a lot of fields. In implementing these algorithms though, application specific issues need to be addressed such as preconditioning for the linear system solution and scattering computation for electron flight for semiconductor device simulation.

1.3.2 TCAD Applications

Since peak performance can only be translated into sustained performance by software, it is the goal of this dissertation to design and implement process and device simulation algorithms suitable for MPPs. MPP algorithms should possess a high degree of parallelism and low communication for good performance. The performance degradation of the parallel algorithm compared to the optimal sequential algorithm should also be compensated by the added computational power.

In the area of the Monte Carlo device simulation, Sugino et al. [26] presents a device simulator which partitions the particles to each processor such that each node in has the same number of particles and the initial spatial distribution of the particle inside the device must be the same for each node. Hiroki et al. [27] investigates load balancing further by sorting the particles according to the three events of free-flight, boundary, and scattering. The particles at each event are then evenly distributed among the processors. This improved the boundary and scattering routine load balance but has little impact on reducing the total computation time. Ranawake et al. [20] describes a parallel Poisson solver and Monte Carlo simulator. Both shared memory and distributed memory algorithms were presented. For the distributed memory, the spatial domain of the device was mapped onto separate processors and dynamic load balancing was implemented to maintain approximately the same number of particles and grid points per processor. Sheng et al. [21] studies the applicability of SIMD architectures for Monte Carlo simulation. The problem is decoupled into two disjoint domains - particle-based and spatially-based. Since the CM-2 supports multiple communication configurations, the simulator can dynamically switch between domains and at the same time minimize interprocessor communications. SIMD Monte Carlo performances can be improved through more efficient flight time generation algorithms which take into account the underlying architecture [22].

Parallel three-dimensional rectangular grid drift-diffusion device simulation algorithms have been designed and implemented by Wu et al. [23], Webber et al. [8], and Tomacruz et al. [9]. Wu et al. [23] presents the device simulator STRIDE which uses incomplete LU decomposition conjugate gradient squared algorithm to solve asymmetric matrices. Cubic partitioning and an incomplete Nested Dissection ordering described in Lucas et al. [28] are used to implement an efficient preconditioner on a MIMD machine. The contribution of this dissertation begins with the development of the a new ordering scheme called the partitioned natural ordering [8] which is observed to be suitable for SIMD architectures. The partitioned natural ordering scheme is then extended for MIMD architectures [9]. Multigrid methods useful for parallel machines are also shown. The ideas and results presented in Webber et al. [8] and Tomacruz et al. [9] are described in chapter 2.

Chapter 3 then investigates the feasibility of irregular grid device simulation on MIMD machines. Parallel algorithms that obtain more than 50% efficiency compared to best sequential algorithms are described. This is achieved through the use of geometrical grid node partitioning and ILU with fill-ins preconditioning routines. This work has been published in Sanghavi et al. [13] and Tomacruz et al. [14].

In the area of semiconductor process simulation, Guerrieri et al. [19] presents a massively parallel algorithm for 2-D scattering in optical lithography. The method is equivalent to the time-domain finite-difference method used in electromagnetic scattering simulations. Due to the regular structure of the problem, each grid point is mapped to a CM-2 processor. Calculating the electric and magnetic field requires nearest neighbor communication. Since the evaluation of boundary conditions require more variables than bulk equations, the additional "dummy" processors provide a local scratch memory. A careful allocation of the variables reduces the sequentiality introduced by the boundary conditions since many instructions used to update the fields in the bulk of the domain are also used to compute the boundary condition. A typical simulation domain consists of a 1024 x 512 mesh structure and steady-state is reached after about 30-50 wave cycles which takes about 5 minutes on an 8k CM-2 machine [24]. A 3-D version implemented on the CM-5 has been developed by Wong and Neureuther [25].

Chapter 4 presents parallel algorithms for the post-exposure bake process simulation of chemically amplified resist systems. The simulations involve the accurate modeling of reaction kinetics and diffusion of acid. For the robustness of the solver, implicit time integration schemes are used. This creates large linear systems of equations that are solved efficiently with a parallel machine. 1-D and 2-D case studies are also presented to verify the proposed diffusion models. This work has been published in Tomacruz et al. [10], Zuniga et al. [11], and Newmark et al. [12].

Chapter 5 summarizes the parallel algorithm and application contributions. A framework for the parallel solution of PDEs is constructed. Generic algorithms which may be useful in other fields are highlighted and areas of future work are presented.

References

[1] R. Dutton and Z. Yu, *Technology CAD: Computer Simulation of IC Processes and Devices*, Kluwer Academic Publishers, Massachusetts, U.S.A., 1993.

[2] M. Lundstrom, *Fundamentals of Carrier Transport*, Addison-Wesley, Reading, Massachussetts, 1990.

[3] G. Heiser, C. Pommerell, J. Weis, and W. Fichtner, "Three-Dimensional Numerical Semiconductor Device Simulation: Algorithms, Architectures, Results", *IEEE Trans. on CAD*, Vol. 10, pp. 1218--1230, Oct. 1991.

[4] K. Nishi and J. Ueda, "Technology CAD at OKI," *Technology CAD Systems*, F. Fashing, S. Halama, and S. Selberherr, eds. Wien, Austria: Springer-Verlag, 1993, pp. 255-273.

[5] M. Noell, S. Poon, M. Orlowski, and G. Heiser, "Study of 3-D Effects in Box Isolation Technologies," *SISDEP Proceedings*, pp. 331-340, 1991.

[6] R. Knepper, et. al., "Technology CAD at IBM," *Technology CAD Systems*, F. Fashing, S. Halama, and S. Selberherr, eds. Wien, Austria: Springer-Verlag, 1993, pp. 25-62.

[7] M. Flynn, "Some Computer Organizations and Their Effectiveness," IEEE Trans. on Computers, Vol. C-21, pp. 948-960, Sept. 1972.

[8] D. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, A. Sangiovanni-Vincentelli, "A Massively Parallel Algorithm for Three Dimensional Simulation", *IEEE Trans. on CAD*, Vol. 10, pp. 1201-1209, 1991.

[9] E. Tomacruz, J. Sanghavi, A. Sangiovanni-Vincentelli, "Algorithms for Drift-Diffusion Device Simulation Using Massively Parallel Processors", *IEICE Trans. on Electronics*, Vol. E77-C, pp. 248-254, Feb. 1994.

[10] E. Tomacruz, M. Zuniga, R. Guerrieri, A. Neureuther, A. Sangiovanni-Vincentelli, "3-D Diffusion

Models for Chemically-Amplified Resists Using Massively Parallel Processors", *SISDEP Proceedings*, pp. 109-112, 1993.

[11] M. Zuniga, G. Wallraff, E. Tomacruz, B. Smith, C. Larson, W. Hinsberg, and A. Neureuther, "Simulation of Locally-Enhanced 3-D Diffusion in chemically amplified Resists," *J. of Vac. Sci. and Tech.*, pp. 2862-2866, Nov/Dec 1993.

[12] D. Newmark, E. Tomacruz, S. Vaidya, and A. Neureuther, "Investigation of Proximity Effects for a Rim Phase-Shifting Mask Printed with Annular Illumination," *SPIE: Optical/Laser Microlithography VII*, 1994.

[13] J. Sanghavi, E. Tomacruz, A. Sangiovanni-Vincentelli, "Massively Parallel Device Simulation Using Irregular Grids", *Nupad Proceedings*, 1994.

[14] E. Tomacruz, J. Sanghavi, A. Sangiovanni-Vincentelli, "A Parallel Iterative Linear Solver for Solving Irregular Grid Semiconductor Device Matrices," *Supercomputing '94*, pp. 24-33, 1994.

[15] G. Almasi, and A. Gottlieb, *Highly Parallel Computing*, 2nd ed., The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994

[16] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.

[17] J. Palmer and G. Steele Jr., "Connection Machine Model CM-5 System Overview," *IEEE 4th Symposium on the Frontiers of Massively Parallel Computation*, pp. 474-483, 1992.

[18] J. Beetem, M. Denneau, and D. Weingarten, "The GF11 Parallel Computer," in. J.J. Dongarra, ed., *Experimental Parallel Computing Architectures*, Amsterdam: North-Holland, 1987.

[19] R. Guerrieri, K. Tadros, J. Gamelin, and A. Neureuther, "Massively Parallel Algorithms for Scattering in Optical Lithography," *IEEE Trans. on CAD*, Vol. 10, pp. 1091-1100, 1991.

[20] U. Ranawake, C. Huster, P. Lenders, and S. Goodnick, "PMC-3D: A Parallel Three-Dimensional Monte Carlo Semiconductor Device Simulator," *IEEE Trans. on CAD*, Vol. 13, pp. 712-724, 1994.

[21] H. Sheng, R. Guerrieri, and A. Sangiovanni-Vincentelli, "Massively Parallel Computation for Three-Dimensional Monte Carlo Semiconductor Device Simulation," *SISDEP Proceedings*, Vol. 4, pp. 285-290, 1991.

[22] H. Sheng, R. Guerrieri, and A. Sangiovanni-Vincentelli, "A Generalized Self-Scattering Technique for Monte Carlo Simulation Suitable for SIMD Architectures," *NASECODE X*, pp. 24-25, 1994.

[23] K. Wu, G. Chin, and R. Dutton, "A STRIDE Towards Practical 3-D Device Simulation - Numerical and Visualization Considerations," *IEEE Trans. on CAD*, Vol. 10, pp. 1132-1140, Sept. 1991.

[24] A. Wong, A. Neureuther, "Mask Topography Effects in Projection Printing of Phase-Shifting Masks," *IEEE Trans. on Elec. Dev.*, Vol. 41, pp. 895-902, June 1994.

[25] A. Wong and A. Neureuther, "Examination of Polarization Effects in Photolithographic Masks using Three-Dimensional Rigorous Simulation," *SPIE: Optical/Laser Microlithography VII*, 1994.

[26] S. Sugino, C. Yao, and R. Dutton, "Parallelization of Monte Carlo Analysis on Hypercube Multiprocessors and on a Networked EWS System," *SISDEP Proceedings*, Vol. 4, pp. 275-284, 1991.

[27] A. Hiroki, S. Odanaka, and A. Goda, "Massively Parallel Computation for Monte Carlo Device Simulation," *VPAD Proceedings*, pp. 18-19, 1993.

[28] R. Lucas, K. Wu, and R. Dutton, "A Parallel 3-D Poisson Solver on a Hypercube Multiprocessor," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 442-445, 1987.

[29] W. Giloi, "Parallel Programming Models and Their Interdependence with Parallel Architectures," *Proceedings of Programming Models for Massively Parallel Computers*, pp. 2-11, 1993.

[30] G. Lerman and L. Rudolph, *Parallel Evolution of Parallel Processors*, Plenum Press, New York, N.Y., 1993.

[31] H. Kung, "Gigabit Local Area Networks: A System Perspective," *IEEE Communications Magazine*, Vol. 30, No.4, pp. 79-89, April 1992.

[32] L. Kleinrock, "The Latency/Bandwidth Trade-off in Gigabit Networks," *IEEE Communications Magazine*, Vol. 30, No.4, pp. 36-40, April 1992.

[33] C. Pancake, "The Changing Face of Supercomputing," *IEEE Parallel & Distributed Technology*, pp. 12-15, Nov. 1993.

[34] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, "A User's Guide to PVM Parallel Virtual Machine," *Oak Ridge National Laboratory Report ORNL/TM-11826*, July 1990.

[35] V. Sunderam, G. Geist, J. Dongarra, and R. Manchek, "The PVM Concurrent Computing System: Evolution, Experiences, and Trends," *Oak Ridge National Laboratory Report*, 1993.

[36] R. Martin, "HPAM: An Active Message Layer for a Network of HP Workstations," *IEEE Hot Interconnects II*, Aug. 1994.

[37] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. of the 19th ISCA*, pp. 256-266, May 1992.

[38] J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," *Oak Ridge National Laboratory Report*, Sept. 1993.

[39] J. Sethian, "Computational Fluid Mechanics and Massively Parallel Processors," *Supercomputing '93*,

pp. 74-82, 1993.

[40] T. Anderson, D. Culler, and D. Patterson, *A Case for Networks of Workstations (NOW)*, U.C. Berkeley Computer Science Seminar, 1994.

[41] D. Korn and N. Rushfield, "Washcloth Simulation of Three-Dimensional Weather Forecasting Code," *NYU Ultracomputer Note #55*, Map 1983.

[42] J. Dennis, G. Gao, and K. Todd, "Modeling the Weather with a Data Flow Supercomputer," *IEEE Trans on Computers*, Vol. 33, pp. 592-603, July 1984.

[43] G. Almansi, B. Alpern, C. Berman, J. Carter, and D. Hale, "A Case-Study in Performance Programming: Seismic Migration," *Proceedings, 2nd Symposium on High Performance Computing*, Oct. 1991.

[44] G. Almansi, T. McLuckie, J. Bell, A. Gordon, and D. Hale, "Parallel Distributed Seismic Migration," *Concurrency Practice and Experience*, Vol. 5, pp. 105-131, April 1993.

[45] C. Seitz, "The Cosmic Cube," *Communications of the ACM*, Vol. 28, pp. 22-33, Jan. 1985.

[46] J. Beetem, M. Denneau, and D. Weingarten, "The GF11 Parallel Computer," J. Dongarra, ed., *Experimental Parallel Computing Architectures*, Amsterdam: North-Holland, 1987.

[47] F. Butler, H. Chen J. Sexton, A. Vaccarino, and D. Weingarten, "Hadron Mass Predictions of the Valence Approximations to Lattice QCD," *Physical Review Letters*, Vol. 70, pp. 2849-2852, 1993.

[48] C. Catlett, "In Search of Gigabit Applications," *IEEE Communications Magazine*, pp. 42-51, April 1992.

[49] A. Majumdar and W. Martin, "Parallel Preconditioned Conjugate Gradient Algorithm Applied to Neutron Diffusion Problem," *Transactions of the American Nuclear Society*, Vol. 65, pp. 209-210, 1992.

CHAPTER 2

Rectangular Grid Drift-Diffusion Device Simulation

2.1 Overview

Numerical modeling of semiconductor devices to predict electrical behavior is important for efficient design of new devices. CADDETH [17], MINIMOS [18], and SITAR [19] are sequential 3-D rectangular grid device simulators that have been successfully used for the past decade. However, even with current vector supercomputers, these device simulators still require significant CPU times for the generation of accurate simulation results. In this chapter, parallel algorithms for rectangular grid drift-diffusion simulation are presented. Ordering schemes for preconditioning are developed and tested for parallel execution efficiency and for convergence robustness. Multigrid methods are also investigated to improve the initial guess for the Newton-Raphson solver. Finally, the parallel algorithms are used to simulate silicon pixel detectors.

2.2 Problem Definition

2.2.1 Device Equations

Shockley's paper in 1949 [1] first described the drift-diffusion equations which model the flow of electrons and holes in a semiconductor material. The steady-state drift-diffusion (DD) model has been used in this chapter and is based on the Poisson's equation and the continuity equations for electrons and holes [13].

$$\nabla \bullet (\varepsilon \nabla \psi) = -q(p - n + N_d - N_a) \tag{2.1}$$

$$\nabla \bullet (\mu_n n \nabla \psi - D_n \nabla n) - R_n = 0 \tag{2.2}$$

$$\nabla \bullet (\mu_p p \nabla \psi + D_p \nabla p) - R_p = 0 \tag{2.3}$$

where the dependent variables to be determined are the electrostatic potential, ψ, and the electron and hole carrier concentrations, n and p. Here,μ_n, μ_p, D_n, D_p are respectively, the mobilities and diffusion coefficients for electrons and holes; ε, q, N_d, and N_a are the permittivity, electronic charge, impurity donor density, and impurity acceptor density respectively; R_n and R_p are recombination-generation rates which include the Auger, Shockley-Read-Hall, and impact ionization terms.

2.2.2 Device Structures

2.2.2.1 Need for 3-D Structures

The general 3-D drift-diffusion model was presented by Shockley [2] and Van RoosBroek's [3] in 1950. However, the development of 3-D simulators did not happen until the early 1980s due to the computational and applications requirement. The 1980s signalled the arrival of supercomputing platforms that provided the necessary computational power. At this same time, the small semiconductor device features necessitated 3-D simulations. When current flow is no longer predominant to a plane, 2-D mesh structures are no longer sufficient. For example, as device dimensions shrink for a MOSFET, edge effects which are characterized by nonlinear current trajectories become important. Another class of 3-D effects is device cross talk such as parasitic MOS effects that degrade device performance under high bias conditions. Table 2.1 illustrates several examples along with the CPU times needed for the simulations.

Table 2.1: Examples of 3-D Device Simulation Applications

Device or Physical Phenomena	Mesh Size	Machine	CPU Time	Reference & Comments
EEPROM	35,000	Sun 4/260	570 s/bias point	Linton et al. [14]; poisson only
CMOS Latch-up	56,562	Cray-2	5 h	Heiser et al. [4]
Parasitic MOS	48,190	Multiflow	8 h	Noell et al. [6]
DRAM	25,000	IBM RS/6000	15-30 m/b.p.	Knepper et al. [15]

2.2.2.2 Sample Structures

Several typical device structures, MOSFETs, and BJTs, are used to test the simulator. The MOSFET shown in Figure 2.1 is an n-channel device with a W/L_{eff} of $1\mu m/1\mu m$ with an oxide thickness of $28nm$. The source and drain regions have an impurity concentration of $2 \times 10^{20} cm^{-3}$ and a junction depth of $0.3um$. The substrate impurity is $2.5 \times 10^{16} cm^{-3}$. The bipolar transistor shown in Figure 2.1 is a standard npn device, with a base width of $0.1\ \mu m$. The emitter impurity concentration is $2 \times 10^{20}\ cm^{-3}$, the collector is $10^{20} cm^{-3}$, the active base region is 1.4 x $10^{18} cm^{-3}$, the base contact region is 3 x $10^{19} cm^{-3}$, the buried N+ region is 3 x $10^{19} cm^{-3}$, and the substrate is $10^{16} cm^{-3}$.

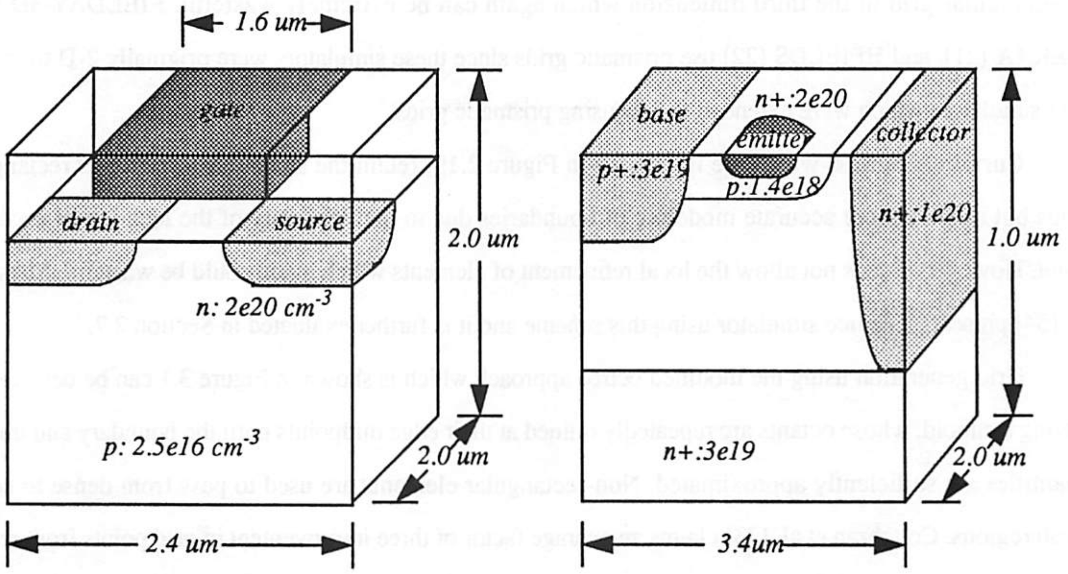

Figure 2.1: MOSFET and BJT Structures

2.3 Solution Method

2.3.1 3-D Grid Generation

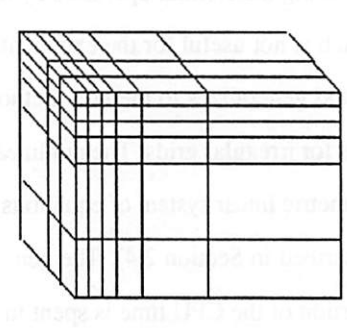

Figure 2.2: Rectangular Grid

Rectangular grids or tensor-product grids which are illustrated in Figure 2.2 are easy to specify but can be wasteful since grid lines may need to be extended to quasi-neutral regions. Device simulators that used this device structures include CADDETH [17], MINIMOS [18], SITAR [19], and STRIDE [9]. The parallel implementation of rectangular grids is the main focus of this chapter.

Prismatic grids can be generated by replicating a triangular grid in the third dimension. This results in a rectangular grid in the third dimension which again can be extremely wasteful. FIELDAY-3D [20], SIERRA [21], and HFIELDS [22] use prismatic grids since these simulators were originally 2-D triangular grid simulators which were extended to 3-D using prismatic grids.

Curvilinear grids, which are illustrated in Figure 2.19, retain the same connectivity as rectangular grids but allow a more accurate modeling of boundaries due to the flexibility of the location of each grid point. However, it does not allow the local refinement of elements which again could be wasteful. Matsuo et al. [54] presents a device simulator using this scheme and it is further evaluated in Section 2.7.

Grid generation using the modified octree approach which is shown in Figure 3.1 can be described as having a cuboid, whose octants are repeatedly refined at their edge midpoints until the boundary and internal quantities are sufficiently approximated. Non-rectangular elements are used to pass from dense to coarse mesh regions. Coughran et al. [23] claims an average factor of three improvement in grid points from rectangular to irregular grids. Also, the non-rectangular elements can be used to approximate arbitrary device surfaces. This is used by the Second [4], Simul [5] and the device simulator presented in Chapter 3.

2.3.2 Drift-Diffusion Solution Method

The DD equations have been discretized on a three-dimensional tensor-product grid using finite difference (FD) [24] and the Scharfetter-Gummel method [25] described by Section 4.3 for the approximation of carrier densities. FD is based on replacing differential operators by difference operators. The unmodified FD only uses linear approximation which is not useful for the exponential variation of the carrier density in the continuity equations. The FD method generalizes to the box method (BM) which is also known as the control volume or finite volume method for irregular grids. The nonlinear equations are solved using a fully-coupled Newton method, and the asymmetric linear system of equations are solved using the Conjugate Gradient Squared (CGS) method [26] (described in Section 2.4). The general computational steps can be summarized by Algorithm 2.1. A major portion of the CPU time is spent in solving linear systems of equations. Hence, the main focus of this chapter is the efficient implementation of a parallel linear solver.

Algorithm 2.1: Steady-State Drift-Diffusion Computational Steps

```
problem read-in and setup
Newton-Raphson loop
    evaluate the equations for the Jacobian and
        right-hand side of the Newton iteration
    solve the associated linear system
post-processing of results
```

2.3.3 Partitioning for Parallelization

There are numerous possibilities for distributing the workload of a device simulator to different processors - grid, time, voltage, and design. Each scheme has its own advantages in terms of ease of implementation and performance improvements. These advantages depend on the algorithms being implemented and also the type of parallel architecture being used. Partitioning in terms of grid points is used for all the parallel algorithms presented in this dissertation. This involves partitioning the mesh structure into subdomains and mapping each subdomain to a processor. Partitioning in terms of grid points offer the most general form of parallelism since it can also be used to do voltage sweep, transient, and design space simulations.

2.3.3.1 Grid Point Partitioning

To map the problem onto a CM-2, each grid point is assigned to a processor. Thus each processor stores the local values of ψ, n, and p, and the three rows of the matrix of the corresponding grid node. Using this allocation, the grid fits naturally on the organization of the machine and, at the same time, variables having a strong coupling due to the spatial adjacency are tightly clustered. The resulting matrix is a banded matrix, whose seven diagonals are 3x3 matrices representing the interaction among the local variables.

The CM-5 mesh structure is divided into rectangular blocks each called a subdomain. Each subdomain is mapped to a processor and the dimensions of the subdomain are powers of 2. The dimensions in each axis are equal or almost equal in order to form cubic subdomains. This minimizes the total surface area which in turn minimizes the data length of communications between processors. A simple row ordering is used to map the subdomains to the CM-5 processors since the fat tree connections allow minimal penalty for communications between arbitrary processors [30].

2.3.3.2 Time Point Partitioning

Han et al. [44] presents a parallel algorithm to solve equations together in parallel at different time-steps. A processor assigned to a later time-step can start working using a good initial condition before the processor assigned to the preceding time-step finishes its computation. The initial condition for the later time step processor is obtained by the quick solution of a coarse grid. Details of the implementation are further discussed by Han et al. [44]. The algorithm was implemented on a MIMD parallel machine and was tested on a GaAs MESFET device. A 13.7 speed-up is observed for a 16 processor computer.

Tai et al. [45] describes a 2-D device simulator that is parallel in both time and space. The simulator was implemented on a SIMD architecture in which the processors are partitioned to different time points. Each group of processors solves a 2-D device structure for a particular time point. A group of processors assigned to a later time-step can start working using a good initial condition before the group of processors assigned to the preceding time-step finishes its computation. The initial condition for the later time step group of processors is provided by the current solution of the group of processors for the previous time point. An explicit method is used to do numerical time integration. A speed-up of 8 is observed for an SOI application.

2.3.3.3 Voltage Point Partitioning

Allowing each processor to solve a voltage point for a voltage sweep simulation is another way to partitioning the device simulation problem. However, the Newton algorithm is known to perform well when a good initial estimate of the solution is given. Hence, a parallel voltage partitioning cannot take advantage of a usually good initial guess obtained by a projection from two previous solutions.

2.3.3.4 Design Point Partitioning

Different geometry features or doping profile values are needed to be simulated in optimizing device performance or in studying the sensitivity of a device characteristics. Sensitivity analysis is important for predicting worst case device performance due to variations in the geometry and doping profile of the device. These variations result from the limitations in the accuracy of semiconductor processing technology. Hence, each of these geometry and doping features can be simulated in parallel.

2.4 Linear System Solution

2.4.1 Iterative Solvers

There are two basic ways to solve a linear system of equations - direct and iterative. Direct solvers are usually preferred because of their reliability and predictability. They are usually based on variants of Gaussian elimination. They construct a lower triangular matrix L and an upper triangular matrix U such that $LU = A$. L and U are also sparse but usually much denser than the original matrix. Nonzeros in L and U appearing in zero positions of the original of the matrix are called fill-ins. The amount of fill-ins increases superlinearly with the problem size and the dimensionality of the problem. The number of fill-ins for a 3-D discretization is higher than for a 2-D discretization with the same number of grid points. The combination of these two factors makes the storage requirements an issue when switching from 2-D to 3-D models.

Demmel et al. [31] describes parallel implementations for sparse matrix direct solvers. Sparse matrix factorization offers more opportunities for exploiting parallelism beyond those available with dense matrices. However, it may be more difficult to attain good efficiency in the sparse case. The main challenge is developing a row ordering algorithm and a matrix to processor mapping algorithm such that the number of fill-ins is minimized and parallelism is maximized. One of the earliest parallel algorithms which is presented by George et al. [32] maps a group of columns to each processor. Clever row ordering may create columns that may be eliminated in parallel. Sequential iterative solvers are observed to be significantly more efficient compared to sequential direct solvers for 3-D device simulation problems. In fact, due to memory and computational requirements, large problems can only be solved using iterative methods for current sequential computing platforms. It will be shown in Section 3.4.7 that even with 100% efficiency for parallel direct matrix solvers, parallel iterative solvers are still more efficient. However, parallel direct solvers may still be useful for very ill-conditioned matrices that may arise for severely biased semiconductor devices.

Matrix splitting methods such as Jacobi, Gauss-Seidel, and Successive Over-Relaxation (SOR) do not play a significant role in the solution of linear systems in device simulation. Splitting matrices usually do not meet the requirements for convergence. Another class of iterative solvers involves the minimization of a convex function. These solvers are composed of routines for generating the search direction and for finding the minimum in the current search direction. The simplest method is the steepest descent solver. It uses the negative gradient at the current position as its search direction. Unfortunately, the speed of convergence may be relatively slow for a relatively flat steep-sided valley. The algorithm is forced to traverse back and forth across the valley rather than down the valley since the gradient directions for each iteration are too similar.

To avoid the pitfalls of steepest descent, search direction generation algorithms that take into account previous search directions have been created. For example, Generalized Conjugate Residuals (GCR) [34] selects a new search direction based on the current residual plus a linear combination of the previous search directions. Due to storage requirements and computational increase as the number of iteration increases, GCR is restarted at regular intervals. Instead of throwing away previous search directions when restarting, the same amount of memory can be saved by throwing the oldest search direction. This approach is called truncation and the truncated version of GCR is called Orthomin [36]. Further improvements in terms of finding the minimum of the current search direction have created the Generalized Minimal Residual Method (GMRES) [35]. These solvers still fail for typically ill-conditioned device matrices [4,33].

The only successful algorithms for device matrices come from the family of biorthogonalization methods which includes Biconjugate Gradients (BiCG) [37] and its variants. BiCG not only solves the primal linear system but also the dual linear system which is composed of the matrix transpose. BiCG is usually able to solve device matrices where restarted GMRES or truncated Orthomin fail. Two famous variants of BiCG are the Conjugate Gradient Squared (CGS) [26] which is described by Algorithm 2.3 and the Bi-CGSTAB [38]. CGS reformulates BiCG with the absence of transposed matrix operations. It is able to achieve this by squaring the update formulas for the residual and the search direction. By setting the residual function of CGS as a function of the squared update formulas, a more convergent solver is observed. The reason is the "contraction effect" which can be described informally as follows: The *k*th combined residual can be written as a product of a polynomial $\Phi_k(A)$ and the initial residue. Assume BiCG converge in the residual, that is, $\|r_k\|$ is smaller than $\|r_0\|$. Hence, the polynomial $\Phi_k(A)$ contracts $\|r_0\|$. It is then possible that $\Phi_k(A)$ contracts $\|r_k\|$ as well. As a result, the "contraction effect" of $\Phi_k^2(A)$ applied to $\|r_0\|$ is expected to be stronger than that of $\Phi_k(A)$. Search direction routines can then be generated consistent with the matrix polynomial With device matrices, a speed-up of two is observed.

Bi-CGSTAB is based on the same matrix polynomials as CGS, but instead of being squared, this polynomial is premultiplied by another polynomial which is based on the steepest descent. The second polynomial damps the effect of divergence in the BiCG polynomial. Pommerell and Fichtner [39] present a comparison of BiCG and its variants.

The BiCG method and its variants require the same basic linear algebra operations - inner products, vector updates, matrix vector products, and preconditioning. Vector updates are trivially parallelizable since each processor updates its "own" segment for a grid point partitioning illustrated by Section 2.3.3.1. Only the inner products $(x \bullet x)$, matrix vector products (Ax), and possibly the preconditioning $(U^{-1}L^{-1}A)$ would require communication calls. The inner products can easily be parallelized. Each processor computes the inner product of two segments of each vector. The results are sent to other processors in order to be reduced to the required global inner product. Matrix vector products require the need for communication to acquire the elements of the vector. Since only nearby nodes are connected, processors are only required to communicate with a few other processors. After parallel communications calls, all computations can efficiently be done in parallel. The preconditioning part is often the most problematic in terms of parallel implementation due to its sequential nature. Section 2.4.2 discusses this problem in detail. Pommerell [40] presents a detailed analysis of iterative methods relevant to device simulation. Tong [41] gives a comparative study of Lanczos methods applied to other applications.

Jones et al. [57] [58], and Demmel et al. [31] present successful implementations of parallel iterative linear solvers. Jones et al. [57] [58] describes a parallel preconditioned conjugate gradient applied to matrices arising from finite element models. The papers conclude that increase in parallelism generated by coloring-based orderings more than offsets any increase in the number of iterations required for the convergence of the conjugate gradient algorithm. Demmel et al. [31] reports several approaches to obtain parallelism in preconditioning. Most publications emphasize the importance of ordering for preconditioning of matrices which is very problem dependent. This dependency will be further examined in the next subsections. Also, research work has also been done on rearranging computational steps for data locality, reduction of synchronization points, and improved overlapping of communication and computation [59].

2.4.2 Preconditioning

The convergence behavior in solving $Ax = b$ to a given accuracy using an iterative solver depends heavily on the problem under consideration. All the methods converge in one single step if the matrix is an identity matrix. Preconditioning transforms the original linear system to $\tilde{A}\tilde{x} = \tilde{b}$ such that $\tilde{A}$ is close to the characteristics of the identity. Hence, one can expect that an iterative method will solve the preconditioned system in fewer iterations than the original system. This approach is useful if the total time to perform all the transformations and the needed preconditioned iterations is smaller than the time for an unpreconditioned solution.

$\tilde{A}$ is calculated by multiplying A with its approximate inverse A^{-1} and $\tilde{b}$ is obtained by multiplying b with A^{-1}. The usual way of obtaining A^{-1} is to factorize A into components that are easily invertible. One choice is LU-factorization, a variant of Gaussian Elimination which computes the lower triangular L, and the upper triangular U such that $A = LU$. A common preconditioner is incomplete LU (ILU). Algorithm 2.2 describes ILU decomposition which is equivalent to LU decomposition if all the fill-ins in step 4 are included. Fill-ins are defined as matrix entries in A that are zero but are not equal to zero in L or U. Since A is only an operator for matrix-vector multiplication in BiCG and its variants, the preconditioner is applied by the following order of operations - $(U^{-1}(L^{-1}(Av)))$ where v is a vector (Algorithm 2.3 illustrates a ILU preconditioned CGS). Using $\tilde{A}v$ may seem more efficient but calculating $\tilde{A}$ is difficult since it is not sparse.

Preconditioning is essential for the convergence of BiCG and its variants for device matrices [39]. Between no preconditioning and full LU, there is a spectrum of preconditioners that offer a wide variety of efficiency-robustness trade-offs. The design of the preconditioner addressed in this chapter requires parallelization considerations in addition to the robustness issues. For a parallel preconditioner, we would like to minimize the total CPU time needed by the parallel machine to complete the solution of the linear systems.

The parallel algorithm uses a variation of the incomplete LU factorization for preconditioning since ILU is observed to be the most efficient for device matrices [39]. The ordering of the equations for this factorization has a significant effect on the convergence behavior of the CGS algorithm as well as on the number of operations that can be carried out in parallel. Unfortunately, more parallelism yields in general slower convergence, so that finding an ordering that minimizes the overall running time is not trivial.

There are numerous orderings of nodes possible [27]. Two criteria are used to examine the orderings in the next subsections. First, the preconditioner should make the iterative algorithm converge for typical ill-conditioned device matrices. Second, it should be able to minimize the total elapsed time to obtain the solution. Processor utilization is correlated to the second criterion but is not the primary concern in performance evaluation.

Algorithm 2.2: ILU Preconditioning

ILU decomposition:

```
Given:    $A$, a nonsingular $n \times n$ matrix.
Step 0:   Set $k = 1$.
Step 1:   Set the $k^{th}$ row of U equal to the $k^{th}$ row of the matrix A.
          $u_{kj} = a_{kj}$ ;   $j = k, k+1, \ldots, n$
Step 2:   If $k = n$ then stop.
Step 3:   Obtain the kth column of L.
          $l_{ik} = a_{ik} / u_{kk}$ ;   $i = k+1, \ldots, n$
Step 4:   Update A. Fill-ins may be introduced.
          $a_{ij} = a_{ij} - l_{ik}u_{kj}$ ;   $i, j = k+1, k+2, \ldots, n$
Step 5:   Increment $k$ and return to step 1.
```

Forward substitution:

$$y_k = b_k - \sum_{j=1}^{k-1} l_{kj} y_j \; ; \quad k = 1, 2, \ldots, n$$

Backward substitution:

$$x_k = \frac{\left(y_k - \sum_{j=k+1}^{n} u_{kj} x_j\right)}{u_{kk}} \; ; \quad k = n, n-1, \ldots, 1$$

Algorithm 2.3: Preconditioned CGS

```
Let:
$r_0 = r = U^{-1}L^{-1}(b-Ax_0)$, $\rho = 1$, $p = 0$, $q = 0$
While ($r \bullet r > \varepsilon$) {
    $\beta = 1/\rho$ ;    $\rho = r \bullet r_0$ ;    $\beta = \beta\rho$ ;
    $u = \beta q + r$ ;    $v = \beta p + q$ ;    $p = \beta v + u$ ;
    $v = U^{-1}L^{-1}Ap$ ;    $\sigma = v \bullet r_0$ ;    $\alpha = \rho/\sigma$ ;
    $q = -\alpha v + v$ ;    $v = u + q$ ;    $u = U^{-1}L^{-1}Av$ ;
    $r = r - \alpha u$ ;    $x = x + \alpha v$
}
```

2.4.3 Ordering of Nodes for the CM-2

2.4.3.1 Natural Ordering and Red-Black Ordering

A commonly used ordering for preconditioning on sequential machines is the natural ordering. This has been successfully used by the device simulator CADDETH [17]. In this ordering, grid nodes (hence matrix equations) are numbered first in the x direction, then in the y direction, and finally in the z direction (or some other permutation of x, y, and z). Specifically, the index for the equation at grid point i, j, k is given by the formula $i + N_x j + N_x N_y k$ where N_x, N_y, and N_z are the number of grid points in the x, y, and z directions respectively.

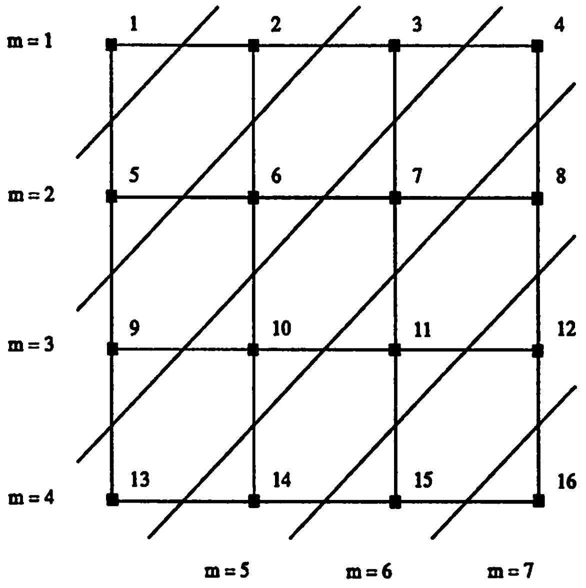

Figure 2.3: Two Dimensional Grid with Natural Ordering

During the LU factorization and the forward and backward substitution, the condition that determines whether or not a node can be eliminated is that all adjacent nodes that have a lower index must have been eliminated already. At first glance this is a completely sequential algorithm. On closer inspection however, a significant number of operations can be carried out in parallel as shown in Figure 2.3 where a two dimensional example is shown. The sets of equations that can be processed in parallel form diagonal lines passing through the grid. In this example, these sets are: {1}, {2, 5}, {3, 6, 9},{4, 7, 10, 13}, {8, 11, 14}, {12, 15}, and {16}. In three dimensions, these sets form diagonal planes. The number of steps required by the algorithm in three dimensions on a massively parallel computer is $N_x + N_y + N_{z-2}$, which is $O(N^{1/3})$ if the number of grid nodes is increased uniformly in each of the three dimensions which is $O(N^{1/3})$ if the number of grid nodes is increased uniformly in each of the three dimensions.

A preconditioner using the natural ordering has good convergence behavior, but it does not exploit well the architecture of a massively parallel processor. In fact, the time per iteration, $O(N^{1/3})$, grows fast with the number of grid points.

A Red-Black ordering [28] provides for a much more efficient parallel implementation of the matrix operations, but as we shall see later, the convergence of the CGS method is significantly slower. This ordering labels each grid node either red or black such that each node is not adjacent to any node of the same color. Thus, given the assumption of no fill-ins, each red node is independent of every other red node, and each black node is independent of every other black node. With this ordering, during the LU factorization, forward and backward substitutions, all of the red nodes can be eliminated simultaneously first, followed by all of the black nodes also simultaneously.

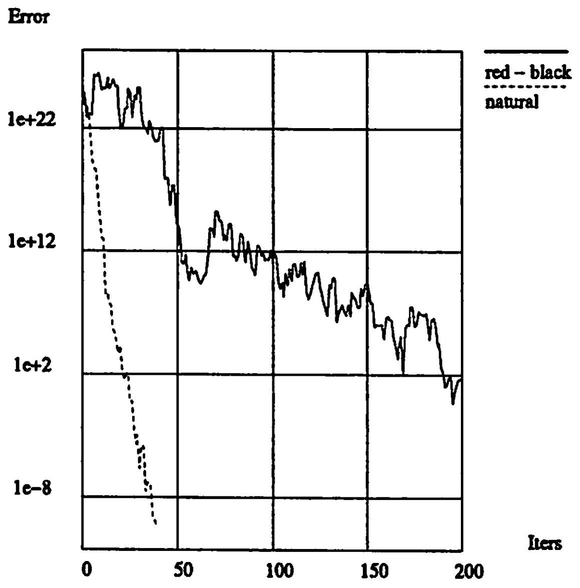

Figure 2.4: Convergence of Natural and Red-Black Ordering

This ordering is very efficient for parallel computers since it needs constant time per matrix iteration, and is the method of choice for solving Poisson's equation [29] where the speed of convergence of the conjugate gradient method is not strongly affected when compared to the natural ordering. However, for the solution of the coupled drift diffusion equations, the CGS method with the red-black ordering converges much slower than the natural ordering as shown in Figure 2.4 for a MOSFET simulation on a 16x16x2 grid. The error is defined to be the maximum residual generated by the CGS algorithm. The total time for solving a matrix using the red-black ordering is larger than that for the natural ordering even though the latter requires much more time per iteration.

2.4.3.2 Partitioned Natural Ordering

Given that the natural ordering performs well for CGS, we examined a modification to the basic scheme to yield a new method that would allow more parallel operations, the partitioned natural ordering. In the natural ordering, the sets of nodes that can be eliminated in parallel form diagonal planes through the grid as shown in Figure 2.3 and a plane has to wait for the factorization process to terminate on the preceding plane. Now if we ignore the dependency between the nodes on plane $m+1$ and the nodes on plane m, then we partition the original matrix into two parts: the one from plane 1 (the left-top node) to plane *m* and the other from plane *m* to plane *n*, the last plane (the right-bottom node) of the grid. In this case, at the first step of the iteration, all the nodes on planes *1* and *m+1* can be processed in parallel. After this step, all the nodes on planes *2* and *m+2* can be processed in parallel, and so on. Using the example in Figure 2.3, the groups of nodes that can be eliminated in parallel are now: {1, 8, 11, 14}, {2, 5, 12, 15}, {3, 6, 9, 16}, and {4, 7, 10, 13}.

This method was first applied to the incomplete LU factorization, but gave disappointing results since many iterations were needed to achieve convergence. However, note that the CGS method requires only one LU factorization per matrix solution, but several forward and backward substitutions (one of each per iteration). Thus it is more important to speed up the forward and backward substitution processes than the factorization process. Hence, the partitioned natural ordering method was applied to forward and backward substitution as follows.

The incomplete LU factorization is carried out using the natural ordering and then entries in the L and U factors that link the elements of the matrix corresponding to the nodes across the partition, are discarded. In this way, the numerical values of the entries of the LU factors are the same as in the natural ordering, but the results of the forward and backward substitution process are different.

If the point at which the planes are partitioned is the same for both the forward and backward substitutions, there would be no data transferred between the partitions because all of the matrix elements connecting the partitions would be set to zero. Therefore the point at which the planes are partitioned is offset for the backward substitution compared to the forward substitution. For example, referring to Figure 2.3, if the partition for the forward substitution is between planes *4* and *5*, then for the backward substitution it would be between planes *3* and *4*.

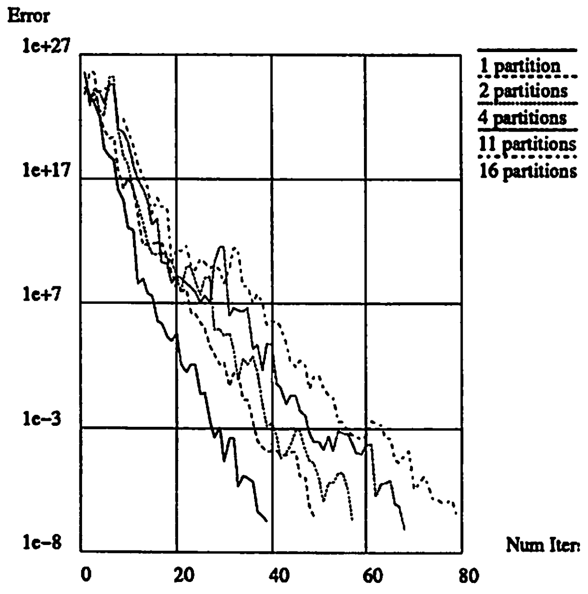

Figure 2.5: Effect of Number of Partitions to Number of Iterations

Of course, this technique is not limited to two partitions. Figure 2.5 shows how the convergence is affected by increasing the number of partitions. As can be seen, the speed of convergence is slower as the number of partitions is increased, but not by orders of magnitude. Figure 2.6 shows the CPU time for the complete solution of a matrix with respect to the number of partitions using this algorithm. The matrix being solved was generated from a MOSFET simulation on a 16x32x8 grid, so the number of sets of equivalent nodes in the dependency graph (or planes through the grid) is 54. This graph shows that the CPU time is minimized when the number of partitions is in approximately the range of 15 through 27, or, in other words, when there are approximately two or three planes per partition. As the number of partitions increases beyond half the total number of planes, the number of iterations for convergence increases significantly. All of the results presented in this paper related to the partitioned natural ordering use two planes per partition since this partition offers the maximum amount of parallelism without affecting the convergence speed by an intolerable amount.

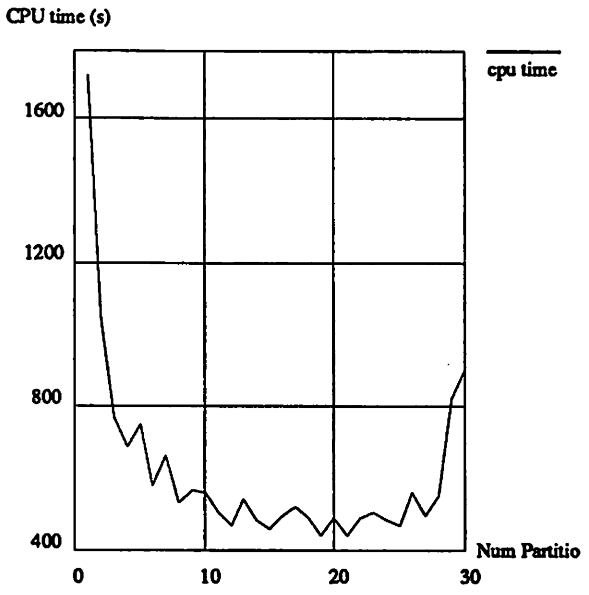

Figure 2.6: Effect of Number of Partitions to Total CPU Time

Since the partitioned natural ordering uses a fixed number of planes per partition (two), the CPU time per matrix iteration is now a constant, as it was with the red-black ordering, but at the same time retains most of the convergence properties of the full natural ordering.

Even with this approach to speed up forward and backward substitutions, it is observed that the time for the LU factorization (even though it is computed with the full natural order) is always significantly smaller, and growing at a smaller rate, than the time for the forward and backward substitutions.

2.4.3.3 Three-Color and Nested Dissection

The two-color ordering labels each grid node either red or black such that each node is not adjacent to any node of the same color. The equation to do incomplete two-color LU decomposition is shown as follows.

$$\begin{bmatrix} L_{RD} & L_{RO} \\ L_{BO} & L_{BD} \end{bmatrix} = \begin{bmatrix} I & 0 \\ A & I \end{bmatrix} \begin{bmatrix} I & B \\ 0 & E \end{bmatrix} + error \qquad (2.4)$$

Since the fill-ins only occur in the *E* part of the *U* matrix, the black nodes should be divided into more groups to allow fill-ins to influence *E*. The simplest approach is to divide the black nodes into two groups, *B1* and *B2*. The same rule used to create red and black nodes can be used to create *B1* and *B2* nodes by assuming red nodes do not exists. This produces a three color ordering and the equation for incomplete LU decomposition is shown as follows.

$$\begin{bmatrix} I_R & A_{RB1} & A_{RB2} \\ A_{B1R} & I_{B1} & 0 \\ A_{B2R} & 0 & I_{B2} \end{bmatrix} \equiv \begin{bmatrix} I_R & 0 & 0 \\ L_{B1R} & I_{B1} & 0 \\ L_{B2R} & L_{B2B1} & I_{B2} \end{bmatrix} \begin{bmatrix} I_R & U_{RB1} & U_{RB2} \\ 0 & E_{B1} & U_{B1B2} \\ 0 & 0 & E_{B2} \end{bmatrix} \tag{2.5}$$

L_{B2B1} and U_{B1B2} are the fill-ins that are supposed to improve the preconditioner. Table 2.2 shows that while the three color scheme improved the quality of the preconditioner, the CPU time required did not make it competitive overall with the simpler red-black ordering.

Table 2.2: 16x32x8 MOS Three-Color Results

Method	CPU Time (relative)	Total Linear Iterations
R/B BCG	1.8	2388
R/B CGS	1.9	3846
R/B1/B2 CGS	13	3544
Natural CGS	1	104

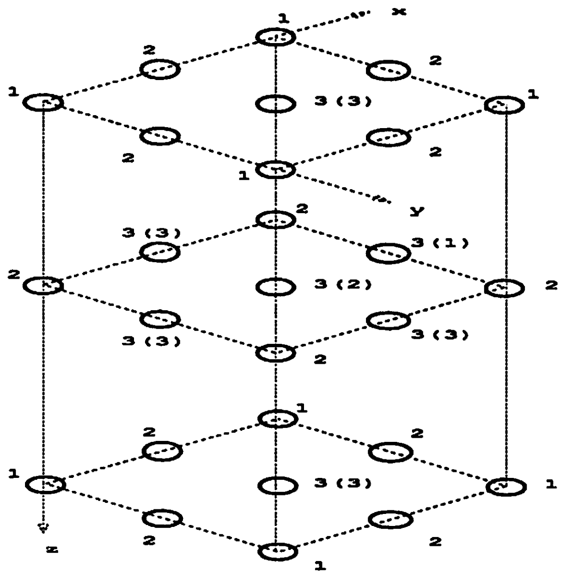

Figure 2.7: Nested Dissection Ordering

The nested dissection ordering [42] has been used for a parallel direct solution of sparse matrices [43]. This ordering minimizes the number of dependencies between nodes which in turn should minimize the number of discarded fill-ins for a preconditioner. The basic idea is to recursively split the mesh structure into two pieces of roughly equal rectangular blocks and a separator plane. A 2-D nested dissection scheme is applied to the separator plane and red-black ordering is used for the nodes that do not belong to any separator sets. The red and black nodes are mapped to the top of the linear matrix. Then, the separator grid points are allocated their corresponding rows in the matrix. The red and black nodes can be individually executed in parallel. The separator nodes can also be executed in parallel using a parallel row ordering scheme. This algorithm is applied to a 3x3x3 mesh structure and the results are shown in Figure 2.7. All nodes labelled with the same number can be executed in parallel except for nodes labelled 3. Group 3 nodes are further labelled with numbers enclosed in parenthesis. Nodes with the same number can be done in parallel.

Current results for nested dissection do not show significant improvements for diode simulations. In fact, MOS simulations run into convergence problems. It is observed that better results are obtained when the coupling of nodes is maintained instead of minimizing the discarded fill-ins.

2.4.4 Ordering of Nodes for the CM-5

2.4.4.1 Partitioned Natural Ordering

The partitioned natural ordering [8] has been extended to the MIMD MPP CM-5 architecture. This ordering compares favorably with other well known techniques such as the red-black ordering and the natural ordering on a SIMD CM-2. The CM-2 and CM-5 implementations are tested with a 32x32x16 MOS transistor and results are presented in Table 2.3 for a single bias point. Generating the linear matrix is relatively inexpensive for both CM-2 and CM-5 implementations. The linear solvers take more than 95% of the CPU time for both implementations. The CM-2 version spends more time in the preconditioning step (incomplete LU factorization or ILU) since this sequential step forces the CM-2 to have more computational resources idle compared to the CM-5.

Table 2.3: CPU Time Breakdown

Operation	CM-2	CM-5
Jacobian	2.0%	3.3%
ILU	30.0%	10.1%
CGS	68.0%	86.5%

Table 2.3: CPU Time Breakdown

Operation	CM-2	CM-5
CGS matrix times vector	26.8%	36.3%
CGS Forward/Backward Substitution (CGS-FBS)	33.4%	42.2%
CGS-FBS Communication	12.1%	15.4%

2.4.4.2 Block Partitioned Natural Ordering (BPNO)

A new ordering scheme tailored for the CM-5 called the *block partitioned natural ordering* is introduced. To allow each processor of the CM-5 to execute in parallel, each subdomain mapped into each processor should be disconnected from other subdomains while doing forward and backward substitution. Using the idea of not having the same cut points for the forward and backward substitution proposed by Webber et al. [8], a new preconditioner called the block partitioned natural ordering preconditioner is proposed. This preconditioner still cuts the links at the boundary of subdomains for forward substitution as shown in Figure 2.8. However, the cut planes for the backward substitution is moved by an offset of one which is illustrated by Figure 2.9. Natural ordering backward substitution is done consecutively from set 1 to set 4. Set 2 is composed of two planes of nodes, set 3 is composed of three lines of nodes, and set 4 has one node. Doing simple subdomain partitioning for backward substitution would have disconnected the set 4 node from its three neighbor subdomains by processing it first. This partitioning gave poor results. The offset of one allows information to travel from one subdomain to another during the preconditioning of the linear matrix. An offset that cuts through the middle of the subdomain gives similar results.

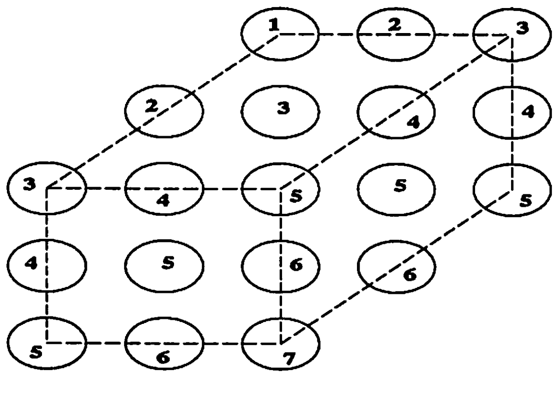

Figure 2.8: BPNO Forward Substitution Subdomain

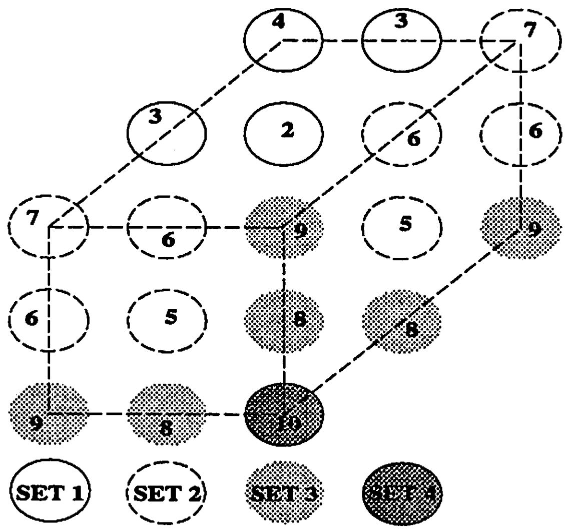

Figure 2.9: BPNO Backward Substitution Subdomain

Fill-ins within each CM-5 subdomain can be included. The architecture and large local memory of the processing nodes of the CM-5 allow several levels of fill-ins within each subdomain while doing incomplete LU decomposition, forward substitution, and backward substitution. Allowing fill-ins only in the incomplete LU decomposition did not improve the linear solver. A significant reduction in the total number of inner loop iterations is observed when fill-ins are also allowed in the forward and backward substitution process.

2.4.5 Results

Results for a bipolar transistor described by Figure 2.1 with V_{be}=0.8 and V_{ce}=1.0 are shown in Figures 2.10 and 2.11. PNO, NO, and BPNO signify partitioned natural ordering, natural ordering, and block partitioned natural ordering respectively. The number attached to BPNO indicates the fill-in levels allowed. 64 processors with no vector units are used for CM-5 simulations and 8k processors with floating point accelerators are used for CM-2 simulations. Fill-ins decreased the total inner loop iterations but not the CPU time. BPNO0 is two times faster than PNO-CM5 for the 32k mesh and produces the lowest CPU time for the CM-5. It is also more robust since PNO does not converge for the 64k mesh. PNO still gives the best performance for the CM-2.

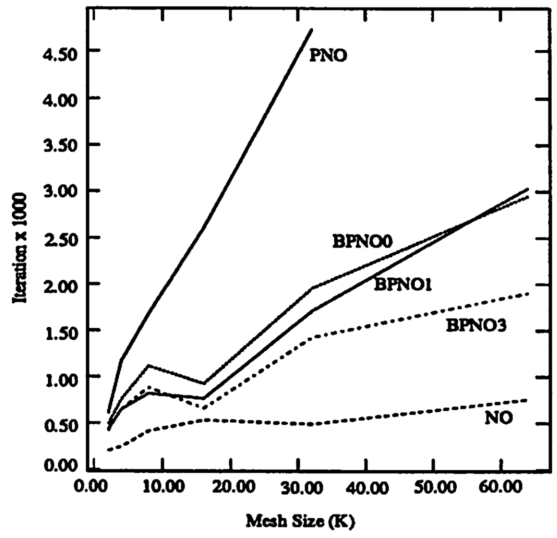

Figure 2.10: Inner Loop (iter) vs. Mesh Size (K)

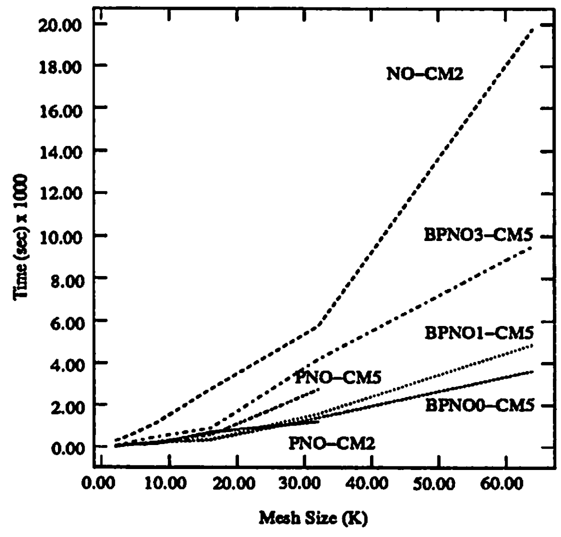

Figure 2.11: CPU Time (sec) vs. Mesh Size (K)

It is stated by Webber et al. [8] that the CM-2 algorithm exceeds vector supercomputer performance for problems greater then 15,000 grid nodes. With reference to Figure 4 which shows comparable performance for the 8K CM-2 and a 64 node CM-5, a 128 node CM-5 can be concluded to provide a vector supercomputer performance. It should be noted that the best known algorithm for each architecture is used for making the performance comparison.

More memory for each CM-5 processor over each CM-2 processor produces a significant improvement in the convergence behavior since more coupling between grid points give better preconditioners. This decrease in inner loop iterations along with faster CM-5 processor elements render a better performance relative to CM-2 type architectures as the problem size gets larger for the proposed algorithms and discretization technique.

Several performance metrics for MPPs discussed by Zorpette [7] are significant for the implemented CM-5 device simulator. The simulator has an 80% to 20% computation to communication ratio which is important in determining which MPP architecture to use and which MPP architectural and software aspects need to be improved. The CM-5 algorithm does not take advantage of overlapping communication and can be executed in a data parallel SIMD mode. These two architectural considerations may become important when nonuniform grids are used. The bisection bandwidth, the rate at which half the processors in the machine can send data to the other half, is important since adjacent subdomains which are arbitrarily mapped to different processors need to communicate with each other. Latency time, the time it takes to prepare for communication, is unimportant because each communication call usually involves transferring hundreds of floating point numbers. Finally, a user specified mapping that takes advantage of certain regular properties in the algorithm and the architecture should give comparable, if not better, performance over MPP architectures that mimic shared-memory.

The CM-5 implementation of the preconditioner can be made more efficient by improving communication and computation routines. For the communication routines, the maximum experimental node to node transfer is found to be 8 Mbytes/sec which is well-below the peak bandwidth of 20 Mbytes/sec [30]. It is expected that this is going to increase with newer versions of the CMMD communications library [30]. For the computation routines, each CM-5 node may have up to four vector units. Each processor can be treated as a 4 processor SIMD computer. Each vector unit is capable of 32 Megaflops/sec. With reference to Figures 1 and 2, all nodes with the same number can be executed in parallel by the vector units.

2.5 Multigrid

The Newton algorithm is known to perform best when a good initial estimate of the solution is given. A good initial guess is usually obtained by a projection from two previous solutions whose bias conditions differ only at one contact to a new applied bias at that contact. The initial guess for the first two solutions may be obtained by the initialization described by Webber et al. [8]. The simulations are initialized with piecewise constant potential and carrier concentration which associate with each region the potential of its contact and the concentration of the majority carrier. This can be accelerated by a multigrid initial guess which does not require any specific knowledge of the device and the region of operation upon which it is simulated. It should be noted that voltage sweeps do not necessarily need to start with zero bias. Measuring threshold voltage or device breakdown for example only involves the simulation of a certain segment of the IV curves. Hence, the first two bias points may significantly affect the total CPU time of voltage segment simulations. The next subsections will present algorithms to obtain the initial bias points efficiently. It may be noted that these algorithms are also applicable to sequential machines.

2.5.1 Multigrid Discretization

The scheme is based on two coarse grid mesh structures intertwined as shown in Figure 2.12. These coarse grids are constructed from 4 sets of discretization nodes. Coarse grid 1 is defined as the union of sets 1 and 2, and coarse grid 2 is defined as the union of sets 3 and 4. Set 1 is defined to be nodes with even coordinate values in all three grid axes, and set 3 is defined to be nodes with odd coordinate values in all three grid axes. Set 2 is defined to be the nodes connecting the nodes of set 1, and set 4 is the nodes connecting the nodes of set 3. If the solution of the equations on one of the coarse grids previously defined is carried out while the other coarse grid mesh structure is used as a boundary condition, the nodes in sets 2 and 4 will have only two active neighbors, thus making possible the static elimination of the variables associated with the nodes themselves. This ultimately allows the use of a smaller grid mesh structure composed solely of set 1 or 3. In addition, the elimination of set 2 or 4 decreases the number of variables which decreases the search space of the linear solver and hence, reduces the number of linear solver iterations.

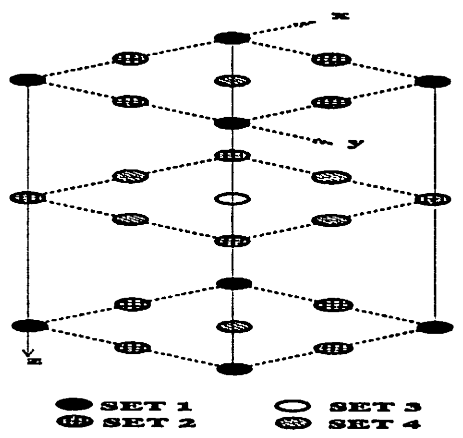

Figure 2.12: Multigrid Discretization

2.5.2 Multigrid Initial Guess

The multigrid discretization is used to define an approximate decoupled Newton scheme to produce a good initial guess for the full Newton algorithm. The algorithms consist of the following steps: First, do a simulation of set 1 nodes. Using set 1 as a boundary condition, an initial guess for set 2 nodes is computed by doing three extended simulations. Each extended simulation uses the actual fine grid discretization for one axis and the set 1 discretization for the remaining two axes. Using sets 1 and 2 as boundary conditions, an initial guess is calculated for sets 3 and 4. As a final step, sets 3 and 4 are used as boundary conditions to improve the initial guess for sets 1 and 2. These simulation steps are summarized as follows (B.C. denotes Dirichlet boundary conditions).

Algorithm 2.4: Multigrid Initial Guess

```
1. Set 1 Simulation (B.C. = 0)
2. Set 2 Simulation (B.C. = Set 1)
    a. extended x
    b. extended y
    c. extended z
3. Sets 3 and 4 Simulation (B.C. = Sets 1 and 2)
4. Sets 1 and 2 Simulation (B.C. = Sets 3 and 4)
```

Simulations with a multigrid initial guess are done for a 31x31x15 MOSFET, a 63x63x31 MOSFET, and a 63x63x31 bipolar transistor described by Figure 2.1 with varying bias conditions using a CM-2. A factor more than 2 in CPU time improvement is observed compared to simulations with initial guess generated by Webber et al. [8] for large and highly biased problems. It is also observed that as the bias conditions of the devices become harder to solve, the multigrid initial guess gives a better relative performance. These results are summarized in Table 2.4 and similar results are expected for the CM-5.

Interpolation schemes were also tried instead of doing a Block Newton Simulation for Steps 3 and 4. Linearly interpolated potential and carrier concentrations did not produce a good initial guess for step 4. The use of Neumann instead of Dirichlet boundary conditions also produced a poor initial guess.

Table 2.4: Multigrid Initial Guess Results

Device	Bias Conditions	CPU Time Speed-up
31x31x15 MOS	Vg=3.0,Vd=5.0	1.3
31x31x15 MOS	Vg=4.0,Vd=5.0	1.8
63x63x31 MOS	Vg=0.5,Vd=0.1	?
63x63x31 MOS	Vg=0.5,Vd=0.5	2.3
63x63x31 MOS	Vg=0.5,Vd=1.0	?
63x63x31 BJT	Vb=0.6,Vc=0.5	?
63x63x31 BJT	Vb=0.6,Vc=1.0	1.5
63x63x31 BJT	Vb=0.6,Vc=3.0	2.5

? = Original simulator did not converge; New simulator converges

2.5.3 Alternating Coarse Grid Simulation (ACG)

An iterative block relaxation Newton can be also defined on the basis of our multigrid discretization. This basically involves looping through steps 3 and 4 of the multigrid grid initial guess routine until a certain convergence criterion is met. It is observed that doing steps 3 and 4 produce well-conditioned linear matrices. This allows the use of partitioned natural ordering incomplete LU decompositions.

Defining convergence as steps 3 and 4 taking only one Newton iteration each to meet the potential convergence criterion, a 10% to 30% CPU time reduction is observed for BJTs. The MOS CPU simulation times did not have any significant reductions.

2.5.4 ACG with Intermediate Selected Nodes Simulation

To improve performance, a simulation of nodes with the largest error is proposed after doing steps 3 and 4. These steps are summarized as follows.

Algorithm 2.5: ACG with Intermediate Selected Nodes

```
1. Set 1 Simulation (B.C. = 0)
2. Set 2 Simulation (B.C. = Set 1)
    a. extended x
    b. extended y
    c. extended z
3. Sets 3 and 4 Simulation (B.C. = Sets 1 and 2)
4. Sets 1 and 2 Simulation (B.C. = Sets 3 and 4)
4.5 Simulate nodes with largest error
5. Repeat steps 3 to 4.5 until convergence
```

Experimental simulations are done to determine which nodes are to be included in the intermediate selected nodes simulation. Poor results are observed for taking the top 12.5% nodes with the largest error. A 2.5 decrease in total final fine grid linear solver iterations is observed for taking the top 1.5% and their neighbors for the intermediate selected nodes.

In implementing a CM-2 version of the scheme presented above, the intermediate selected nodes are proposed to be chosen to fit inside a rectangular mesh structure for easy and efficient mapping into the CM-2 architecture. The top 1.5% plus neighbor is still used as a criteria for the important nodes to simulate. The first selection is done by enclosing in a rectangular grid box the most number of important nodes. Two other variations are done by dividing the covering box into two boxes and into eight smaller boxes. It is observed that the single box is competitive with the 2 and 8 box selection. Efficient routines are still needed to be developed to solve this covering problem. A CM-5 implementation does not require rectangular intermediate selected nodes for efficient simulation.

A rectangular covering heuristic is needed for the CM-2 implementation of ACG with intermediate selected nodes in order to do timing comparisons. A CM-5 implementation of ACG with intermediate selected nodes will require some load balancing heuristics when doing step 4.5.

2.6 Silicon Pixel Detector Application

2.6.1 Background

Silicon pixel detectors are being given more attention in the high energy physics community since they can be used as very effective tracking devices. Pixel Detectors may offer very small detection elements (as small as 30 x 30 um^2) with low capacitance, low leakage current and no ambiguities in multiple-hit events. This implies a very high noise immunity and an intrinsic radiation resistance, which are good features for applications with future high luminosity machines.

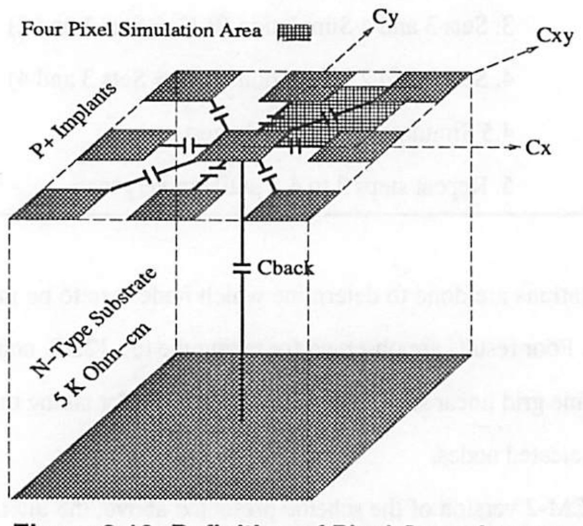

Figure 2.13: Definition of Pixel Capacitances

The pixel capacitances play a crucial role in the system design since they determine the noise and cross-talk of the detector. Capacitances for 100 x 100 um^2 pixel detectors illustrated in Figure 2.13 are measured and simulated. By comparing experimental and simulation results, the pixel capacitances for varying geometries are expected to be modelled. Each pixel is a P+-implant with an aluminum contact on top, laid-out at a fixed 100um pitch with varying gaps (5,10,15,20,25, and 30 *um*). The other side of the implant has an n+-implant that serves as a back contact. The main concern is with the surface capacitances C_x and C_{xy} defined in Figure 2.13. The back capacitance is simulated to be roughly equal to the area of the pixel (pitch2) divided by the silicon thickness (C_{back} = 3.5*fF* for the pixels) with a slight dependence on the gap, similar to the one observed in strip detectors [46].

Section 2.6.2 illustrates the special measurement techniques that had to be developed to account for stray capacitances. Section 2.6.3 describes the method used for calculating capacitances from simulation results and compares the simulation with measured data.

2.6.2 Measurements

The main sources of error in the measurement of small capacitances are the noise in the system and the stray capacitances. Bosisio et al. [47] describes the difficulty and method of the experimental measurements. One interesting observation is that the pixel capacitances lay in the 10 *fF* range and, even with coaxial probes, the residual capacitance of the tips placed at 100 um distance, *Cpp*, is the order of 10 *fF*. Its proper subtraction is crucial for the correct measurement of pixel capacitance and to measure it, the wafer has to be lowered from the contact position as shown in Figure 2.14. Unfortunately, the residual probe tips capacitance depends rather strongly on the separation *d* of the probe tips from the wafer conductive plane.

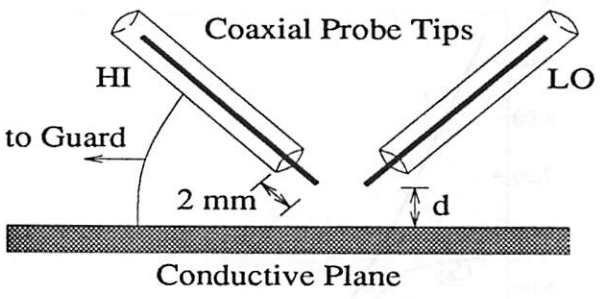

Figure 2.14: Probe Tips Set-up for Cpp Measurement

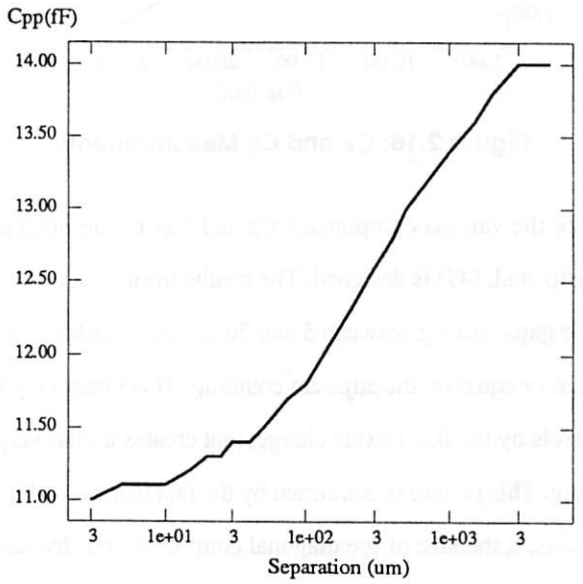

Figure 2.15: Probe Tips Residual Capacitance

Figure 2.15 shows the *Cpp* measurements and it can be interpreted as follows: For large separations (*d* > tip length (2 *mm*)), the conductive plane is unseen and *Cpp* is constant. For medium separations (tip-to-tip distance (100 *um*) < *d* < tip length), the conductive plane is beginning to "eat-up" some of the field lines and the capacitance decreases. The dependence is *Cpp* α *log(d)* as the probe tips are seen at this distance as wires. For short separations (10 *um* < *d* < 100 *um*), the geometry gets more complex because the distance between the tips is comparable with separation. For very short distances (*d* < 10um), *Cpp* is flat again and is basically the capacitance of the probe tips laying on a conductive plane with an infinitely thin insulator layer on top. The *Cpp* measurement is done with *d* equal to 10 *um* which is where the curve is almost flat.

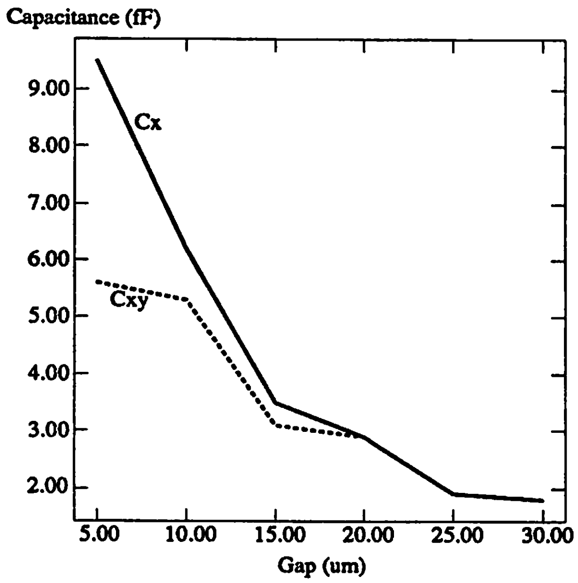

Figure 2.16: Cx and Cy Measurements

To separately measure the various components *Cx* and *Cxy* of the pixel capacitance, a specific test structure described by Bosisio et al. [47] is designed. The results from separate measurements of *Cx* and *Cxy* are shown in Figure 2.16 for gaps varying between 5 and 30 *um*. A remarkable feature of the plot is that the diagonal is of the same order, or equal to, the adjacent coupling. This effect may be due to the accumulation channel created between pixels by the fixed oxide charge, that creates a relatively high conduction path and increases long range coupling. This picture is confirmed by the fact that at small gaps, when the resistance of the accumulation layer increases, the size of the diagonal coupling to *Cx* decreases. If the capacitances are correctly measured, one expects a simple relationship between *Cx*, *Cxy*, and *Cone*: $Cone = 4(Cx + Cxy)$.

This is actually the case in the measurements, as can be seen in Figure 2.17, where the two quantities are plotted on the same graph.

2.6.3 Comparison with Simulations

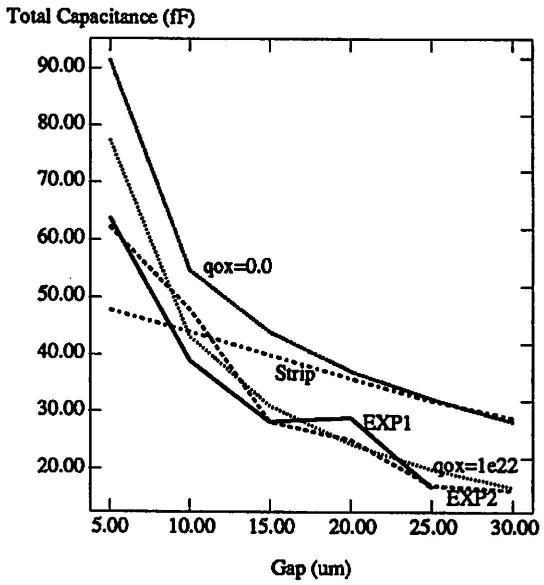

Figure 2.17: Cone from Experimental and Simulation Measurements

The simulation of capacitances in semiconductor devices requires a basic steady state solution for the desired operating condition plus some means of estimating the small signal response of the device. Laux [50] describes three methods to measure the capacitance - charge partitioning (CP), fourier decomposition of transient analysis (FD), and sinusoidal steady state analysis (S3A). The CP method is currently utilized since it has the least computational requirements. In CP method, after the basic solution is found, a voltage step ΔV_j is applied to the electrode *j* and the new, slightly modified DC solution is calculated. The element of the capacitance matrix *Cij* is given by the charge variation on electrode i divided by the voltage step: $Cij = \Delta Q_i / \Delta V_j$. The difficulty lies tin the definition of ΔQ_i, that generally must include not only the charge induced on the contact but also the bulk charge associated with that electrode. For this association to be effective, a device dependent physical insight is needed. For the pixel detectors being described, the charge region boundaries were placed in the middle of the gaps between pixels.

The parallel device simulator described earlier is used to simulate four and nine pixel devices. Different mesh structures are used depending upon the device being simulated - 16x16x16, 32x32x32, and 64x64x64. The comparison of measurements with simulation is shown in Figure 2.17. There are two simulation curves that refer to different values of fixed positive oxide charge. This interface charge [51] is formed during the oxidation process and is localized at the Si-SiO2 interface. Unfortunately the simulation lacks the code to handle a surface charge, and we therefore distribute it in the oxide volume. We added a rough factor of two in the equivalence to take into account the diminished effectiveness on the accumulation layer of a volume distributed charge, so that (volume charge) = 2(surface charge)/(oxide thickness). In any case, although the oxide charge is needed to have a good agreement with measurements, the simulation results show a rather weak dependence on the precise value of the charge. We see from Figure 2.17 that the best agreement is found for $Q_{ox} = 4 \times 10^{11} cm^{-2}$ and in fact direct measurements of the interface charge on MOS capacitors give $Q_{ox} \approx 2 - 4 \times 10^{11} cm^{-2}$.

The simulation of the separate contributions of *Cx* and *Cxy* does not for the moment agree very well with the data due to the uncertainties in the definition of the charge region. As discussed by Ward and Dutton [52] for a MOS transistor, when the charge region boundary lies in an undepleted zone, which is the case of the gap between pixels when the fixed oxide charge is present, its exact placement can become critical and undermine the capacitance simulation. Various algorithms for the charge region boundary placement are under study. The straight line superimposed on Figure 2.17 represents a rough extrapolation from strip detectors data [48] obtained by scaling the interstrip capacitance per unit length C_{IS} with the pixel perimeter. It must be noted that no data exist for strip detectors with very small gaps, and hence, the linear dependence shown by C_{IS} at larger gaps was extrapolated.

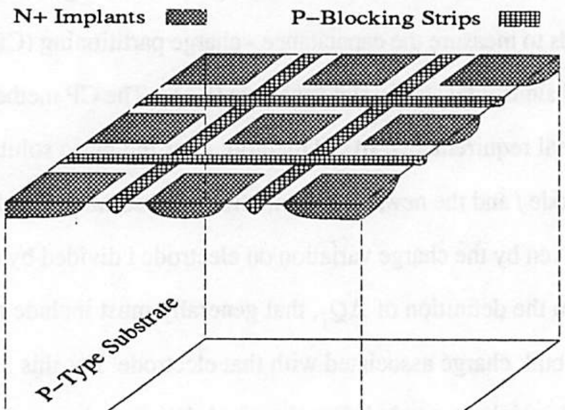

Figure 2.18: P-Type Substrate

To use p-type substrate detectors, it is necessary to put p-blocking strips between each n+ pixel. This prevents the inversion of surface, which is due to the presence of oxide charge, from shorting the n+ regions. Simulations show a factor of 2 increase in total pixel capacitance. A probable reason is the amplification of long range effects due to the blocking strips.

Table 2.5 shows the CPU times required for the simulation of mesh structures with different sizes. Simulations for structures with fixed oxide charge required a voltage ramp for convergence. A 16 x 16 x 16 and a 32 x 32 x 32 simulation is done for the 4-pixel N-Sub device. Only a 5% difference is observed. The P-Sub device simulation only needed two voltage steps. However, the matrices were more ill-conditioned and required the natural ordering preconditioner in some cases for convergence. Nine pixel simulations are also done to see the long range coupling of pixels. Only a 5% increase is observed for the current structures. All simulations are done with a 16K CM-2 equipped with floating point accelerators.

Table 2.5: CPU Times for Silicon Pixel Detector Simulations

Device	Oxide Charge cm^{-3}	Mesh	Voltage Steps	Total Newton Iterations	Total Inner Loop Iterations	Total CPU Time (sec)
4-pixel N-Sub	0	16 x 16 x 16	2	13	1020	98
4-pixel N-Sub	1e22	16 x 16 x 16	27	366	28776	2774
4-pixel N-Sub	1e22	32 x 32 x 32	27	413	103050	19491
4-pixel P-sub	0	32 x 32 x 32	2	37	28673	11040
9-pixel N-sub	0	30 x 26 x 16	2	21	1838	191

2.7 Extensions to Curvilinear Grids

As integrated circuits decrease in size, device simulation techniques which can accurately treat pn-boundaries or shapes of internal boundaries between semiconductors and insulators become important. One limitation of rectangular grids is that they do not allow the exact modeling of nonrectangular device boundaries. Boundary conditions need to be approximated using fine rectangular discretizations which may introduce inaccuracies and redundant grid points. An alternative is the use of curvilinear coordinate systems (CCS) [53] which can model arbitrarily shaped boundaries.

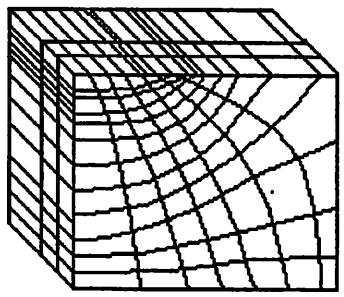

Figure 2.19: Curvilinear Coordinate System

Figure 2.19 illustrates a sample CCS mesh structure. The connectivity between grids is identical to a rectangular mesh. Hence, algorithms presented earlier for parallel rectangular grid device simulation are applicable for this discretization. Instead of using information from 7 points for each control volume, 27 points are necessary for a general CCS. Matsuo et al. [54] presents a simplified version which only requires information from 11 points for each control volume. Both methods can easily be incorporated to in a parallel implementation since the 27 and 11 point discretizations will still only require regular neighbor communication. As shown earlier, this can be easily and efficiently done for parallel processor architectures.

Matsuo et al. [54] presents the simulation of narrow channel effects in MOSFETs using CCS. The increase in threshold voltage (applied gate voltage when drain current is $1nA$) for narrow channels due to boron channel-stop encroachment into the channel region is simulated. Tanaka et al. [55] shows the application of CADDETH-NP [54] to the simulation of a grooved gate MOSFET. The short-channel effects in the sub-0.1-um regime is shown to be suppressed though the use of simulations by a grooved gate MOSFET. This suppression is due to the concave corner of the gate insulator which is extremely grid wasteful to simulate with rectangular grids. Tanaka et al. [56] presents experimental results that confirms the suppression characteristics.

Matsuo et al. [54], and Tanaka et al. [55] [56] use CADDETH-NP for CCS device simulation. The same natural ordering incomplete LU preconditioner used by CADDETH [17] is utilized for these CCS simulations. The matrix components generated from rectangular grids and CCS are similar. Hence, rectangular and CCS matrices should have similar conditioning characteristics. Based on the results presented in Figure 2.10, BPNO with enough levels of fill-ins should be able to converge quickly with CCS matrices. To implement a parallel CCS device simulator from a parallel rectangular grid simulator, modifications of the matrix generator and the matrix-times-vector routines would be needed to handle the additional neighbor connections.

2.8 Summary

3-D simulation is harder than 2-D simulation in several ways. First, the memory requirements grow linearly with the problem size. Second, the CPU time per iteration of the linear solver is proportional to the number of equations. The conditioning of the matrix tends to deteriorate with increasing number of unknowns. Hence, the simulation time grows superlinearly with problem size. Third, more sophisticated visualization programs are needed to examine simulation data.

Three-dimensional device simulations are observed to be very computationally intensive even with vector supercomputers. The main computational task is the solution of the sparse linear system of equations which may have more than a million equations. The efficiency of the iterative linear solver is determined by the preconditioning scheme. The partitioned natural ordering has been developed and published by Webber et al. [8]. It is observed to give the best results for the CM-2 in terms of CPU time minimization. Another contribution is a preconditioner called the block partitioned natural ordering (BPNO) for a CM-5 drift diffusion simulator. BPNO has been published by Tomacruz et al. [60] and it gives an efficient iterative linear solver. It is observed that preconditioners that maintain coupling between nodes give the best results. Also, not having the same cut points for forward and backward substitution is important for producing converging preconditioners.

A multigrid discretization has also been developed to provide a framework to perform a block Newton iteration [60]. Three variations of a block Newton iteration are shown to be effective in generating a good initial guess for the device simulator without having any knowledge of the device structure and the operating region. These schemes are observed to decrease the CPU time by a factor of two.

The parallel algorithms are shown to successfully simulate silicon pixel detectors [47]. Three dimensional capacitance simulations which match experimental results are observed to be significantly different from two dimensional simulations. 3-D long range pixel coupling are observed to be amplified due to the blocking strips.

References

[1] W. Shockley, *Bell System Tech. J.*, Vol. 28, P. 435, 1949.

[2] W. Shockley, *Electrons and Holes in Semiconductors*, Chap. 12, D. Van Nostrand, New York, 1950.

[3] W. Van Roosbroeck, *Bell System Tech. J.*, Vol. 29, p. 560, 1950.

[4] G. Heiser, C. Pommerell, J. Weis, and W. Fichtner, "Three-Dimensional Numerical Semiconductor Device Simulation: Algorithms, Architectures, Results", *IEEE Trans. on CAD*, Vol. 10, pp. 1218--1230, Oct. 1991.

[5] *Simul User's Manual*, Integrated Systems Laboratory, ETH, Zurich, Switzerland, 1993

[6] M. Noell, S. Poon, M. Orlowski, and G. Heiser, "Study of 3-D Effects in Box Isolation Technologies", *SISDEP Proceedings*, pp. 331-340, 1991.

[7] G. Zorpette, "The Power of Parallelism", *IEEE Spectrum*, pp. 28-33, Sept. 1992.

[8] D. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, and A. Sangiovanni-Vincentelli, "A Massively Parallel Algorithm for Three-Dimensional Device Simulation", *IEEE Trans. on CAD*, Vol. 10, pp. 1201-1209, Sept. 1991.

[9] K. Wu, G. Chin, and R. Dutton, "A STRIDE Towards Practical 3-D Device Simulation - Numerical and Visualization Considerations," *IEEE Trans. on CAD*, Vol. 10, pp. 1132-1140, Sept. 1991.

[10] R. Dutton, "Algorithms and TCAD Software using Parallel Computation", *VPAD Proceedings*, pp. 10-12, 1993.

[11] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.

[12] J. Palmer, and G. Steele Jr., "Connection Machine Model CM-5 System Overview", *IEEE 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 474-483, 1992.

[13] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*, Springer-Verlag, Wien, Austria, 1984.

[14] T. Linton, P. Blakey, and D. Neikirk, "The Impact of 3-D Effects on EEPROM Cell Performance," *IEEE Trans. on Elec. Dev.*, Vol. 39, No. 4, pp. 843-850, April 1992.

[15] R. Knepper, et. al., "Technology CAD at IBM," *Technology CAD Systems*, F. Fashing, S. Halama, and S. Selberherr, eds. Wien, Austria: Springer-Verlag, 1993, pp. 25-62.

[16] E. Takeda, K. Takeuchi, D. Hisamoto, T. Toyabe, K. Ohshima, and K. Itoh, "A Cross Section of Alpha-Particle-Induced Soft-Error Phenomena in VLSIs," *IEEE Trans. on Elec. Dev.*, Vol. ED-36, pp. 2567-2575, 1989.

[17] T. Toyabe, H. Masuda, Y. Aoki, H. Shukuri, and T. Hagiwara, "Three-dimensional device simulator CADDETH with highly convergent matrix solution algorithms," *IEEE Trans. Electron Devices*, Vol. ED-32, pp. 2038-2044, 1985.

[18] M. Thurner, and S. Selberherr, "The Extension of MINIMOS to a Three Dimensional Simulation Program," *NASECODE V Proceedings*, pp. 327-332, 1987.

[19] W. Bergner and R. Kircher, "SITAR - An Efficient 3-D simulator for Optimization of Nonplanar Trench Structures," *SISDEP Proceedings*, pp. 165-174, 1988.

[20] E. Buturla, P. Cottrell, B. Grossman, and A. Salsburg, "Finite-element Analysis of Semiconductor Devices: The FIELDAY program," *IBM J. Res. Develop.*, Vol. 25, pp. 218-239, 1981.

[21] J. Chern, J Maeda, L. Arledge, and P. Yang, "Sierra: a 3D Device Simulator for Reliability Modeling," *IEEE Trans. on CAD*, Vol. 8, pp. 516-527, 1989.

[22] P. Ciampolini, A. Pierantoni, M. Melanotte, C. Cecchetti, C. Lombardi, and G. Baccarani, "Realistic Device Simulation in Three Dimensions," *IEDM Proceedings*, pp. 131-134, 1989.

[23] W. Coughran, Jr., M. Pinto, and R. Smith, "Adaptive Grid Generation for VLSI Device Simulation," *IEEE Trans. on CAD*, Vol. 10, pp. 1259-1275, Oct. 1991.

[24] G. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Oxford, 1978.

[25] Scharfetter, D. and Gummel, H., "Large-Signal Analysis of a Silicon Read Diode Oscillator", *IEEE Trans. Electron Devices*, Vol. 16, pp. 64-77, 1969.

[26] Sonneveld, P., "CGS, A Fast Lanczos-type Solver for Nonsymmetric Linear Systems", *SIAM J. Sci. Stat. Comp.*, Vol. 10, pp. 36-52, Jan. 1989.

[27] Duff, I. and Meurant, G., "The Effect of Ordering on Preconditioned Conjugate Gradients", *BIT*, pp. 635-657, 1989.

[28] J. Ortega and R. Voigt, "Solutions of Partial Differential Equations on Vector and Parallel Computers," SIAM Rev., Vol. 27, pp. 149-240, 1985.

[29] R. Guerrieri and A. Sangiovanni-Vincentelli, "Three-Dimensional Capacitance Evaluation on a Connection," *IEEE Trans. Computer-Aided Design*, Vol. 7, pp. 1125-1133, 1988.

[30] Bozkus, Z., Ranka, S. and Fox, G., "Benchmarking the CM-5 multicomputer", *IEEE 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 100-107, 1992.

[31] J. Demmel, M. Heath, and H. van der Vorst, *Parallel Numerical Linear Algebra*, Computer Science Division Tech Report UCB//CSD-92-703, U.C. Berkeley, Oct. 1992.

[32] A. George, M. Heath, J. Liu, and E. Ng, "Sparse Cholesky Factorization on Local-Memory Multiprocessor," *SIAM J. Sci. Stat. Comput.*, Vol. 9, pp. 327-340, 1988

[33] R. Bank, W. Coughran, M. Driscoll, R. Smith, and W. Fichtner, "Iterative methods in semiconductor

device simulation," *Computer Physics Communications*, Vol. 53, pp.201-212, 1989.

[34] H. Elman, *Iterative Methods for Large, Sparse Nonsymmetric Systems of Linear Equations*, Ph.D. Thesis, Yale University, Department of Computer Science, April 1982.

[35] Y. Saad and M. Schultz, "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comput.*, Vol. 7 pp. 856-869, July 1986.

[36] P. Vinsome, "ORTHOMIN - An Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations," *Proc. Fourth SPE Symposium on Reservoir Simulation*, Vol. SPE 5739, pp. 149-160, Feb. 1976.

[37] R. Fletcher, "Conjugate Gradient Methods for Indefinite Systems," *Proc. of the Dundee Biennial conference on Numerical Analysis*, G. Watson ed., Springer-Verlag, New York, 1975.

[38] H. van der Vorst, "Bi-CGSTAB: A Fast and Smoothly Converging Variant of CGS for the Solution of Nonsymmetric Linear Systems," submitted to *SIAM J. Sci. Stat. Comput.*.

[39] C. Pommerell and W. Fichtner, "New Developments in Iterative Methods for Device Simulation," *SISDEP Proceedings*, Vol. 4, pp. 243-248, 1991.

[40] C. Pommerell, *Solution of Large Unsymmetric Systems of Linear Equations*, Ph.D. Dissertation, Swiss Federal Institute of Technology, Zurich, 1992.

[41] C. Tong, *A Comparative Study of Preconditioned Lanczos Methods for Nonsymmetric Linear Systems*, Sandia Report, SAND91-8240, UC-404, Mar. 1992.

[42] George, A. and Liu, J., *Computer Solution of Large Sparse Positive-Definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[43] Lucas., R., Blank, T. and Tiemann, J., "A Parallel Solution Method for Large Sparse Systems of Equations", *IEEE Trans. on CAD*, Vol. 6, pp. 981-991, Nov. 1987.

[44] X. Han, D. Barry, M. Howes, "A Parallel Algorithm for Gummel Iterations in Device Simulation," *Nasecode X Proceedings*, pp. 101-102, 1994.

[45] G. Tai, C. Korman, and I Mayergoyz, "A Parallel-In-Time Method for the Transient Simulation of SOI Devices with Drain Current Overshoots," *IEEE Trans. on CAD*, Vol. 13, pp. 1035-1044, Aug. 1994.

[46] E. Barberis et al., "Capacitances in Silicon Microstrip Detectors," Preprint UCSC, SCIPP 93/16, Aug. 1993.

[47] L. Bosisio, F. Forti, E. Tomacruz, "Measurement and Tridimensional Simulation of Silicon Pixel Detector Capacitance", *IEEE Nuclear Science Symposium*, Vol. 1, pp.338-342, 1993.

[48] K. Yamamoto et al., "Interstrip Capacitance of the Double Sided Silicon Strip Detector," *IEEE Nuclear*

Science Symposium, Orlando Florida, p. 163, 1992.

[49] G. Batignani et al., "Development and Performances of Double Sided Readout Silicon Strip Detectors," *Nuclear Instruments and Methods*, Vol. A310, p. 160, 1991.

[50] S. Laux, "Techniques for Small-Signal Analysis of Semiconductor Devices," *IEEE Trans. on Elec. Dev.*, Vol. ED-32, p. 2028, 1985.

[51] S. Sze, *Physics of Semiconductor Devices*, Wiley, New York, p. 380, 1981.

[52] D. Ward and R. Dutton, "A Charge-Oriented Model for MOS Transistor Capacitances," *Journal of Solid-State Circuits*, Vol. SC-13, p. 703, 1978.

[53] K.Tago, "Semiconductor Device Simulation Method Using Boundary-Fitted Curvilinear Coordinates and the Voronoi discretization," *Electronics and Communications in Japan, Part 2*, Vol. 71, No. 7, pp. 65-76, 1988.

[54] H. Matsuo, J. Tanaka, A. Mishima, K. Tago, and T. Toyabe, "Three-Dimensional Device Simulation with Arbitrary Curved Boundaries Using the Voronoi Discretization Method," *SISDEP Proceedings*, Vol. 4, pp. 157-163, 1991.

[55] J. Tanaka, T. Toyabe, S. Ihara, S. Kimura, H. Noda, and K. Itoh, "Simulation of Sub-0.1-um MOSFET's with Completely Suppressed Short-Channel Effect," *IEEE Trans. on Elec. Dev. Let.*, Vol. 14, No. 8, pp. 396-399, 1993.

[56] J. Tanaka, S. Kimura, H. Noda, T. Toyabe, and S. Ihara, "A Sub-0.1-um Grooved Gate MOSFET with High Immunity to Short-Channel Effects," International Electron Devices Meeting, Washington D.C., pp. 537-540, 1993.

[57] M. Jones, P. Plassmann, "Solution of Large, Sparse Systems of Linear Equations in Massively Parallel Applications," *Supercomputing '92*, pp. 551-560, 1992.

[58] M. Jones, P. Plassmann, "Scalable Iterative Solution of Sparse Linear Systems," *Parallel Computing*, Vol. 20, pp. 753-773, 1994.

[59] J. Lewis and R. van de Geijn, "Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithms," *Supercomputing '93*, pp. 484-492, 1993.

[60] E. Tomacruz, J. Sanghavi, A. Sangiovanni-Vincentelli, "Algorithms for Drift-Diffusion Device Simulation Using Massively Parallel Processors," *IEICE Trans. on Electronics*, Vol. E77-C, pp. 248-254, Feb. 1994.

CHAPTER 3

Irregular Grid Drift-Diffusion Device Simulation

3.1 Overview

The simulation of complex three-dimensional semiconductor devices requires computers with significant computational power. Chapter 2 (published by Webber et al. [1] and Tomacruz et al. [44]), Wu et al. [3] and Dutton et al. [4] have shown how massively parallel computers can be used efficiently for drift-diffusion device simulation. All these simulators used rectangular grids since they are easy to implement, have perfect load balance, and have regular communication patterns. However, irregular grids are important in the field of device simulations since they allow the modeling of nonrectangular device boundaries and do not require grids for quasi-neutral regions. Coughran et al. [5] gives an example of a diagonal alpha particle track that would require 2,000,000 rectangular grid points to model accurately whereas a general irregular grid would only require 6900 grid points to achieve the same accuracy. Even with the reduction of grid points obtained by the use of irregular grids, semiconductor simulation still requires significant computational power. A standard latch-up problem, which requires over 50,000 irregular grid nodes, may take five hours to simulate on vector machines such as the Cray-2 [6]. Other applications such as SOI, parasitic MOSFETs [7], and silicon pixel detectors [8] may require more computational power. Although faster vector supercomputers may offer the computational power needed, parallel processors provide an attractive alternative.

A Connection Machine 5 (CM-5) [9] device simulator that uses an irregular grid automatically generated by the Omega program [10,11] will be presented. For sequential device simulators, the nonlinear algebraic system of equations arising from the discretization is efficiently and accurately solved by a variation of the basic Newton-Raphson algorithm. As usual in this algorithm, most of the computation time is spent on the solution of the linearized system of equations. The focus of the work is to speed-up this step. Heuristics for partitioning, communication scheduling, and preconditioning for the efficient implementation of a parallel iterative linear solver will be illustrated. Parallel results are compared with a sequential program called PILS [12].

This chapter is organized as follows. An overview of the device equations and how they are generally solved is first given. Section 3.3 describes a parallel linear solver. Results are provided and analyzed in Section 3.4. Finally, possible extensions are described in Section 3.5.

3.2 Problem Definition and Solution Method

The steady-state drift-diffusion model of semiconductor devices described by Equations 2.1 - 2.3 has been used in this chapter. The box method (BM) [16] along with the Scharfetter-Gummel method [15] described by Section 4.3 is used to obtained the discrete equations. The solution method for these discrete equations is summarized by Algorithm 2.1. BM uses Gauss's theorem to equate the divergence of a vector field to a scalar source term. It is equivalent to the finite difference method for rectangular grids. BM produces coupling between different grid points only if the points are neighbors. Hence, the linear system resulting from the Newton scheme is very sparse with about 10 nonzeroes per row. Each grid node maps into three rows of the matrix. Because of grid irregularity, the sparsity structure is complex when compared to a simple band structure arising from standard partial differential equations.

For spatial discretization, the grid generator Omega [10,11] which refines mesh density according to geometry and gradient of impurity concentration is used. Omega also computes the cross sections perpendicular to each edge which are required for integrating the device equations with the box method. The desired mesh density inside the device is obtained through recursive refinement of prisms, pyramids, and cuboids. For example in refining along one, two, or three coordinate axes, the cube is subdivided into two halves, four quadrants, and eight octants, respectively. The elements with additional edge midpoints are then subdivided into tetrahedra, pyramids, prisms, or bricks. Sample mesh structures are illustrated by Figures 3.1 - 3.2. A CM-2 mapping in which each irregular grid is mapped to a processor would result in a very inefficient device simulator since CM-2 non-neighbor communication routines are in the order of milliseconds. Taking advantage of near neighbor communication through ingenious mappings is difficult since the difference between the smallest and the largest node degrees is more than ten. Hence, only a CM-5 implementation is developed since the CM-5 does not require near neighbors for efficient communication.

Heiser et al. [6] states that for a typical sequential 3-D irregular grid simulation with a number of grid points of the order of 100,000, between 70 and 90% of the total simulation time is used to solve linear systems. Hence, for the current implementation, linear device matrices are currently generated with the sequential program called Second [6]. Matrix generation is easily parallelizable since it only requires neighbor information and its optimization is not important since it is only executed once for each Newton iteration.

3.3 Static Grid Algorithms

3.3.1 Preconditioned Linear System Solution

The CGS [17] iterative solver which has been presented in Section 2.4.1 is again used for the solution of the linear system. As discussed in Section 2.4.1, inner product, matrix times vector, and preconditioning operations may require interprocessor communication. A communication scheduling algorithm (presented later in Section 3.3.4) based on a graph algorithm is now used to handle the irregularity of communication calls. The same scheduling routine may be used for preconditioning purposes.

The irregularity of the mesh structure makes the use of block partitioned natural ordering (BPNO) described in Section 2.4.4.2 difficult. Keeping track of subdomain boundaries and moving the subdomain boundary is not trivial to implement. Hence, we investigate simplified preconditioners generated by the following combination of ILU preconditioning and FBS techniques: parallel ILU / parallel FBS, sequential ILU / parallel FBS, and sequential ILU / sequential FBS. The parallel version ignores communication completely as opposed to its sequential counterpart. The reverse Cuthill-McKee, minimum degree, and maximum degree algorithms [19] are implemented for ordering processor and grid nodes within each processor. The reverse Cuthill-Mckee ordering starts with the node with the least edge degree. It then selects the neighboring nodes in ascending edge degree order. The algorithm repeats the same process for each neighbor until all the nodes have been ordered. The minimum degree ordering arranges the nodes in ascending edge degree order. The maximum degree does the order in a descending manner. No particular ordering is followed when there are nodes with equivalent edge degrees for all three orderings.

3.3.2 Partitioning

3.3.2.1 Complexity of the Objective Function

The goal is to minimize the total elapsed time needed to obtain the solution. The total elapsed time is determined by the processor that has the longest CPU time requirement to carry out its alloted computation and communication task. Hence, the goal of a partitioner is, for a given algorithm, to achieve computational balance, to achieve communication balance, and to minimize communication requirements.

To achieve computational balance in conjugate gradient based iterative solvers, there are three issues. First, the linear algebra operations between synchronization points must be computationally balanced by balancing the number of grid nodes. These synchronization points which require processor to processor communication are the dot product and matrix times vector operations. Second, balancing the matrix times vector operation is dependent upon the number of grid points and the sum of node degrees. Finally, balancing the application of the preconditioner is a function of the sum of node degrees in a reachability graph for a given node ordering and level of allowed fill-ins.

Two issues need to be addressed to achieve communication balance and to minimize the total communication requirement for a given algorithm. First, the average and maximum numbers of processor neighbors need to be minimized. Processor neighbors are defined as the number of processors a specific processor needs to communicate with to accomplish its computational task. Minimizing processor neighbors would imply minimizing the latency time required for each communication call. Second, the average and maximum numbers of edges cut for each processor need to be minimized.

There are three operations that may require communication for a CG based solver - dot product, matrix times vector, and application of preconditioner. Dot product communication can be done easily through the use of the CM-5 reduction operation. To minimize the number of edges cut and the processor neighbors for a matrix times vector operation, we only have to look at the actual connectivity mesh structure. For preconditioner communication, we have to look into the edges of the reachability graph that go from one processor to another. This reachability graph is dependent upon the given node ordering and the criterion used for allowed fill-ins.

3.3.2.2 Simplified Objective Function

It is difficult to devise an objective function for a partitioner with a CG based solver as its target algorithm when dealing with all the issues described in the previous section. We simplify the criterion for the partitioner by focusing on the following parameters with the actual mesh as the basis graph:

1. time to run the partitioner
2. difference between the processor with the most nodes and the processor with the least nodes (node load balance)
3. average number of processor neighbors
4. maximum number of processor neighbors
5. total edges cut
6. difference between the processor with the most edges and the processor with the least edges (edge load balance)

These parameters will have different significances for the unpreconditioned CG based solver and for the type of preconditioner used. We investigate the geometrical, topographical, and spectral partitioning heuristics that recursively bipartition [20] the graph representing the input mesh structure. Each partitioner has strengths and weaknesses based on the parameters presented above. The significance of each parameter can be determined by the total elapsed CPU time results for each partition.

3.3.2.3 Geometrical Partitioner

The geometrical partitioner, described by Algorithm 3.1, sorts the grid points according to their coordinates along the axis that has the most number of unique coordinate points. Several criteria can be used to choose the partition point.

1. node with a coordinate different from its neighbor closest to the middle
2. node at the middle of the list
3. node that divides the list into two partitions with an equal sum of node degrees

For criterion 2, the grid points at the boundary (grid points with the same coordinates as the node at the middle of the list) are further sorted using the other two axes. This enables most adjacent boundary grid points to remain in the same partition.

Algorithm 3.1: Geometrical Partitioner

Step 1:	For the given set of points, create three sorted list in terms of coordinates for each axis.
Step 2:	Use the list with the most number of grid points and create two groups of points based on a specific partitioning criteria.
Step 3:	Repeat recursively.

The partitioner based on the first and second criteria can be enhanced further by swapping nodes between the two partitions to obtain sum of node degrees balance. More heuristics may be used to choose the region in which to allow the swapping of nodes. The basic idea is to first swap nodes near the boundary and gradually move in until the desired sum of node degrees balance is obtained.

3.3.2.4 Topographical Partitioner

The Fiduccia-Mattheyses algorithm [21], which is an improvement of a local search algorithm first presented by Kernighan and Lin [22], is used to implement a topographical partitioner. The basic algorithm moves a node from one partition to the other partition in an attempt to minimize the cutset while maintaining the load balance between the two partitions within a specified tolerance. A cutset is defined as the number of edges that connect nodes in different partitions. The algorithm has a cost function that is allowed to increase in order to help the partitioner get out of a local minima. Algorithm 3.2 summarizes the topographical partitioning steps. Ten trials are executed for each partitioning result in order to desensitize the partitioning outcome from the random initial guess. A 5% maximum load balance deviation tolerance is used for each binary partition.

Algorithm 3.2: Topographical Partitioner	
Step 1:	Random partitioning of grid points into two groups.
Step 2:	Switch grid points between the two groups. Keep track of total edges cut at each step.
Step 3:	Use partition with lowest number of total edges cut.
Step 4:	Repeat recursively.

3.3.2.5 Spectral Partitioner

The spectral partitioner [23] described by Algorithm 3.3 is based on the computation of the second largest eigenvalue and the corresponding eigenvector of the Laplacian matrix of the connectivity graph. The connectivity graph is the device mesh structure. This eigenvector which is called the Fiedler vector [24] gives distance information about the nodes. Sorting the nodes according to this information provides a way of partitioning the mesh.

Algorithm 3.3: Spectral Partitioner	
Step 1:	Compute Fiedler vector for graph using Lanczos algorithm [24].
Step 2:	Sort vertices according to size of entries in Fiedler Vector
Step 3:	Assign half of the vertices to each subdomain
Step 4:	Repeat recursively.

3.3.3 Domain to Processor Mapping

No particular order is followed in mapping each mesh subdomain to a processor. The partition number generated by the partitioner for each subdomain is also used as the processor number. The fat tree network of the CM-5 minimizes the maximum distance between processors [25]. Hence, we expect minimal communication penalty for using a simple domain to processor mapping.

3.3.4 Communication Scheduling

Given a partition of grid nodes among processors, the abstract model used for scheduling communication is a processor-communication graph. Each vertex represents a processor and each edge (i,j) represents the existence of at least one mesh edge that has grid nodes in processors i and j. Therefore, an edge in processor-communication graph represents communication either in one direction (preconditioner computation, forward substitution, and backward substitution) or in both directions (matrix-vector multiplication).

The communication scheduling algorithm for matrix-vector multiplication is based on the repeated application of the maximal nonbipartite matching heuristic. We decided to use a heuristic algorithm instead of an exact minimizing algorithm since the heuristic algorithm is significantly faster to execute and the heuristic solution is comparable to the exact minimized solution. Each pass of the matching heuristic selects the maximal number of communication edges such that no two have a common vertex. At each step of the maximal matching heuristic, an edge that connects a vertex of maximum degree to its neighbor with largest degree is extracted as a matched edge and deleted from the graph along with the matched vertices and their incident edges. Therefore, this greedy matching heuristic tries to minimize the number of passes while attempting to maximize the number of matched edge during each pass.

The communication scheduling algorithm for the preconditioner computation that needs to send the rows of a matrix from one processor to another is implemented by a simple heuristic that receives data from and sends data to neighboring processors in a specific order. The same heuristic is used for scheduling communication for the forward and backward substitution.

3.4 Experimental Results

3.4.1 Data Structures and Code Optimization

A hash table is used to store each row of the matrix and corresponding vectors. A doubly linked list of two-dimensional dense blocks is used to represent each row. Arrays are used to transfer data from one processor to another.

The performance of each CGS iteration is improved by several modifications. First, the removal of subroutine calls avoids unnecessary memory loads and stores. Second, CGS loop reorganization, which is done by moving some independent CGS steps, improves the cache performance. Third, the combination of several linear algebra operations minimizes memory loads and stores. Fourth, address precomputation speeds-up memory access. Application of all these modifications provides a 4X speed-up for CGS with no preconditioning and a 3X speed-up for CGS with preconditioning.

3.4.2 Computing Environment & Test Examples

All results in Sections 4.4 and 4.5 are obtained with a 64 node CM-5 with no vector units. 32, 64, and 128 node CM-5s with no vector units are used to generate the results in Section 3.4.6. The algorithms are implemented using the C programming language with the CMMD 3.0 communications library [26]. Only blocking communication calls which prevents the overlap of computation and communication are currently used.

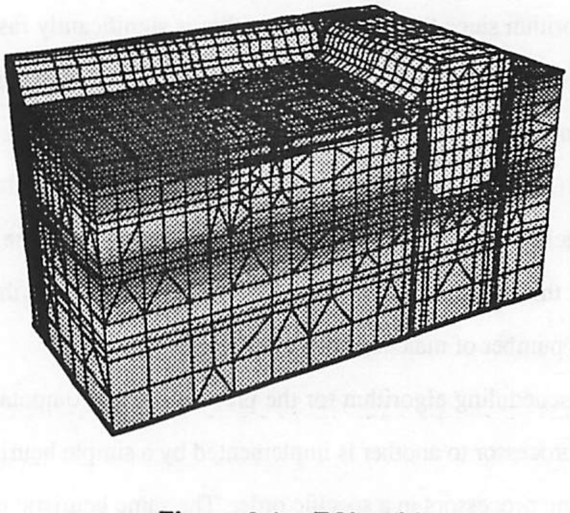
Figure 3.1: ECL grid

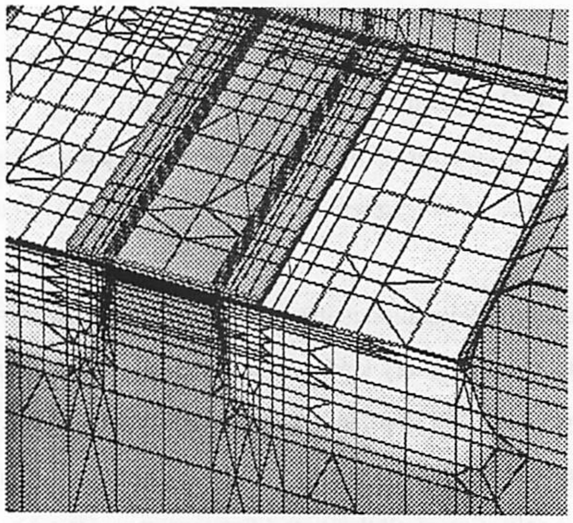
Figure 3.2: LOCOS grid

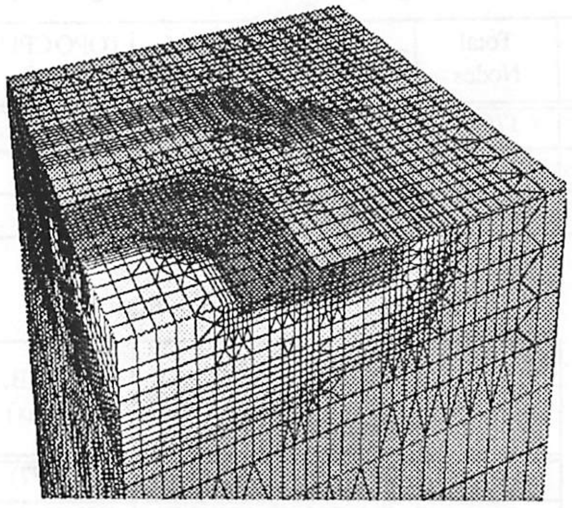
Figure 3.3: MCT Grid

Several device structures described by Hitschfeld [27] are used to study the partitioning schemes described above. ECL is a trench-isolated bipolar transistor, LOCOS is a short channel MOS transistor with surrounding locos isolation, and MCT is a MOS-controlled Thyristor with integrated MOS controlled n+ emitter shorts and a bipolar gate. It is a device used in high power applications such as traction, high-voltage transmission and motor control. Varying bias conditions and initial guess values are used to test the solver.

3.4.3 Partitioning Results

In terms of single Sparc partitioning CPU time shown in Table 3.1, both topographical and spectral partitioner are an order of magnitude slower than the geometrical partitioner. Criterion 1 in Section 3.3.2.3 is used for the geometrical partitioning results in this section. The topographical partitioner can be accelerated by decreasing the number of random initial guesses. However, this significantly degrades partitioning results. Table 3.2 illustrates that the spectral partitioner gives the best node load balance (L.B.). By using Criterion 2 in Section 3.3.2.3, the geometrical partitioner will give perfect node load balance. The topographical partitioner can also be modified to give good load balance by tightening the node load balance tolerance in the cost function.

Table 3.1: Single processor partitioning time (sec)

Device	Total Nodes	Total Edges	GEOM CPU Time	TOPO CPU Time	SPEC CPU Time
ecl	17678	58794	72.4	1722.7	793.6
locos	16586	57335	67.0	1492.0	1250.6
mct	41122	140529	178.0	7486.4	Failed

Table 3.2: Load balance results

Device	GEOM L.B. min (max)	TOPO L.B. min (max)	SPEC L.B. min (max)
ecl	208 (339)	205 (396)	276 (277)
locos	210 (308)	252 (267)	259 (260)
mct	548 (736)	625 (666)	Failed

Processor neighbors (P.N.) are defined as the number of processors a processor needs to communicate with while doing a matrix times vector operation that is basically represented by the mesh connectivity graph. The geometrical partitioner is shown by Table 3.3 to give the best minimization of average and maximum number of processor neighbors. This results degrades when the cost function is adjusted to improve node load balance. The topographical partitioner is illustrated by Table 3.4 to give the lowest number of edges cut for the three sample devices. The number of edges cut increases when the cost function is adjusted to improve the node load balance. The total number of edges cut for the geometrical and the spectral partitions may be improved by applying a variation of the topographical algorithm. Grid nodes at partition boundaries may be swapped to decrease the number of edges cut and, at same time, still maintain the node load balance. In terms of the difference between processors with the most edges and the least edges, the three partitioners gave comparable results. The sum of node degrees has a worst case deviation of 30% from the average.

Table 3.3: Processor neighbor results

Device	GEOM Ave., Max.	TOPO Ave, Max	SPEC Ave, Max
ecl	8.0, 17	8.5, 17	10.5, 19
locos	7.9, 14	9.8, 15	11.2, 19
mct	7.1, 14	8.0, 20	Failed

Table 3.4: Edges cut results

Device	GEOM E.C.	TOPO E.C.	SPEC E.C.
ecl	9112	7749	11641
locos	9294	8155	12223
mct	15396	13601	Failed

We focused on the geometrical partitioner since it can be easily modified and executed to produce partitions with varying characteristics. The topographical and spectral partitioners are also examined to see if the best results obtained with the geometrical partitioner can be improved.

3.4.4 CGS with No Preconditioning

With regards to the simplified cost function, there is a trade-off between load balance and communication balance. Comparable results were obtained with different partitioning goals - node load balance, edge load balance, and minimization of processor neighbors. The only major performance degradation observed was while using the partitioning with the perfect node and edge balance. This partition resulted in a severe number of processor neighbors and of total edges cut. Hence, it can be concluded that the performance of CGS with no preconditioning is relatively insensitive to the type of partitioning.

A variation from 10 to 40% of the total CPU time is currently spent on communication. The variation is due to the fact that there is a computational imbalance between communication calls, imbalance of processor communication requirements, and idle time due to the difference in processor neighbors. Minimizing total edges cut by using the topographical partitioner did not improve the performance.

A speed-up of more than 40X for 100 CGS iterations with no preconditioning is obtained for large problems with the geometrical partitioner. This speed-up corresponds to an efficiency of more than 60% in obtaining the theoretical maximum speed-up. The results are shown in Figure 3.4. The same speed-up is obtained for a 41,122 node MCT. It should be pointed out that PILS is a highly optimized sequential solver and the parallel code is also optimized as described in the previous section.

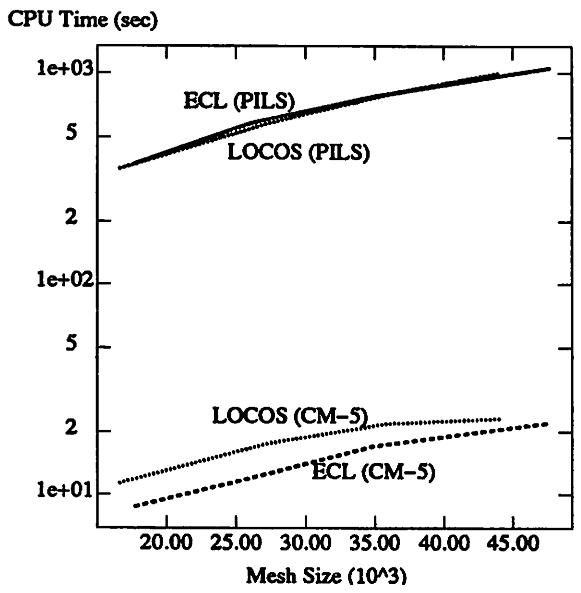

Figure 3.4: CGS with no preconditioning results

3.4.5 CGS with Preconditioning

3.4.5.1 ILU with Magnitude Threshold Fill-ins

In implementing the preconditioner on the CM-5, we observed that any variation of the sequential FBS produced unsatisfactory results even with different variations of ordering schemes. Since FBS is applied twice in each iteration, its sequentiality left a major portion of the processors idle. Hence, we decided to use a parallel FBS which did not require any communication and focused on improving the matrix decomposition routine. The parallel LU (PLU) preconditioner obtained by performing a complete LU decomposition within each processor is found to be robust in practice and completely parallel since communication between processors is now eliminated. However, PLU is very computationally expensive in the calculation of L and U. This problem is alleviated by introducing an incomplete LU routine with some allowed fill-ins. The fill-ins are kept or discarded depending upon their magnitude. A sequential version called ILUV has been shown to be effective on the hydrodynamic equations by Zhao et al. [28].

Gauss's algorithm [29] is used to generate the incomplete LU decomposition. Fill-ins are generated in Step 4 of the algorithm. For a coupled solution, a fill-in unit is a three by three matrix and is kept if at least one of the entries is larger than the specified threshold.

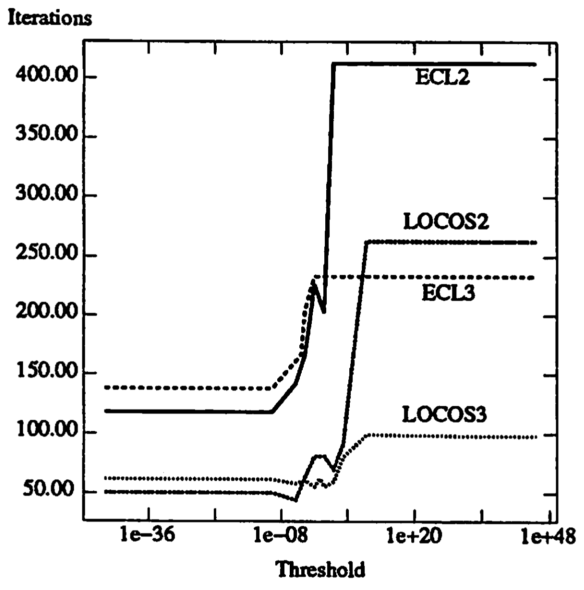

Figure 3.5: Effect of fill-ins on number of iterations

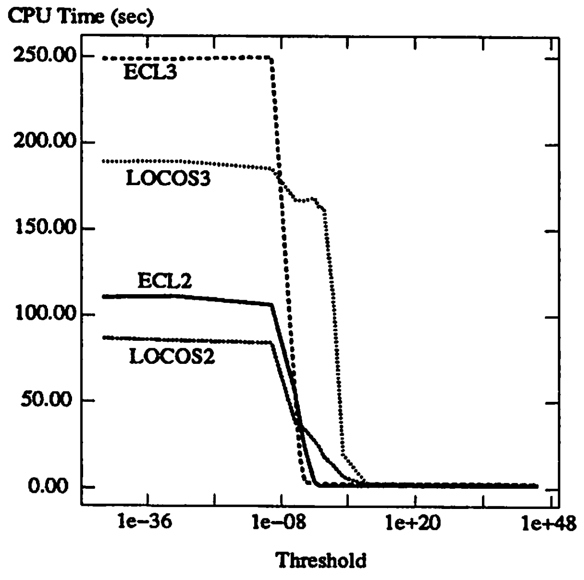

Figure 3.6: ILU CPU time

Figure 3.5 illustrates the influence of the number of fill-ins on the number of iterations needed for convergence. ECL2, ECL3, LOCOS2, and LOCOS3 are 25969, 34877, 27288, and 35875 node devices respectively. Figure 3.6 shows how much faster it is to compute L and U when less fill-ins are retained. Figure 3.7 shows how threshold levels influence the total CPU time which is the composed of ILU computation and of the total time needed for CGS to converge. Optimal results show a factor of more than 35X speed-up compared to PILS. However, differences in the conditioning of the matrix require different threshold levels for optimal performance.

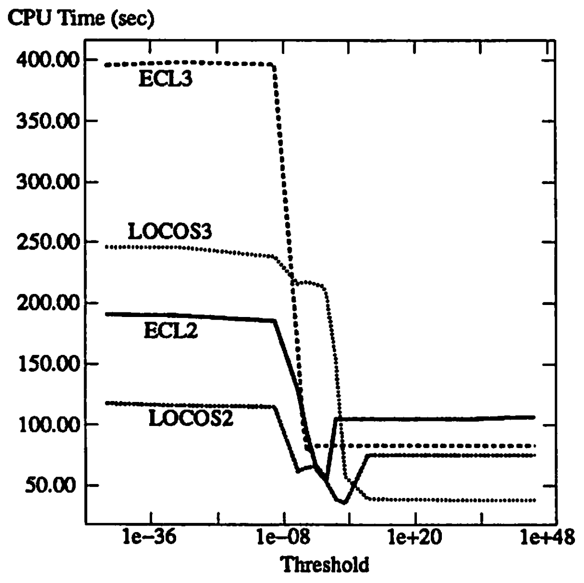

Figure 3.7: Total CPU time

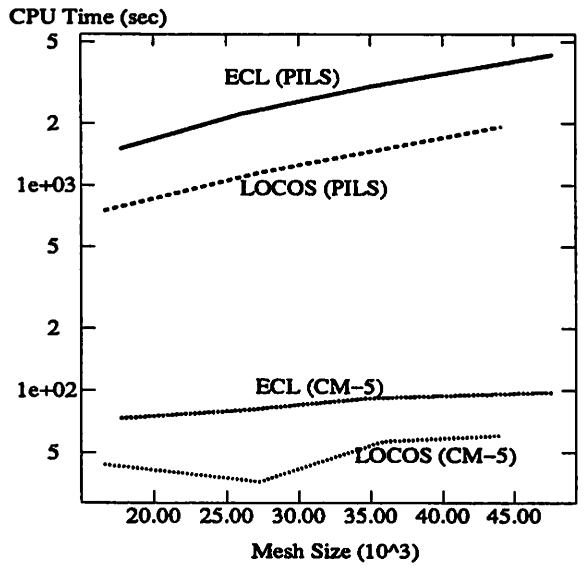

Figure 3.8: Sequential and parallel results

3.4.5.2 Automatic Selection of Threshold

Figure 3.7 shows that for large problems, no fill-ins are needed to obtain the optimum performance. However, for smaller problems, optimum performance is observed with fill-ins. We implement a routine that automatically searches for the threshold that will create a number of fill-ins comparable to the number of entries in the matrix being solved. The threshold search is done using a bisection method. Matrix decomposition is aborted at early stages of the computation if the current fill-in count predicts a substantially lower or higher final count compared to the number of blocks in the A matrix.

Figure 3.8 show that results of automatic threshold selection algorithm and of the sequential PILS solution. Sequential results are obtained using CGS with an ILU preconditioning which is commonly used in sequential device simulation. The time to do the incomplete factorization for PILS is less than 2% of the total CPU time to solve the linear system of equations. Automatic selection of the threshold is also used for larger problems and minimal CPU time penalty is observed. Sequential to parallel speed-up increases as the problem size increases. Speed-ups of 50% of the theoretical maximum are observed for large problems. Similar speed-up is obtained for the MCT device.

3.4.5.3 Effects of Partitioning

For simulations with little or no fill-ins, the best results are obtained with partitions having perfect node load balance. Since there is no increase in communication compared to CGS with no preconditioning, a variation from 2 to 10% of the total CPU time is spent on communication. The same reasons discussed in Section 4.4 explain this variation.

For simulation with fill-ins, an increase in problem size increases the computational imbalance for FBS. Balancing the number of grid points per processor is not sufficient for computational balance. For example, in the 25969 node ECL device with 400 nodes for each processor, there is a worst-case 27/35 discrepancy in terms of sum of node degrees. This produced a 49/85 discrepancy in the number of fill-ins generated by the LU decomposition within each processor. This imbalance produced a factor of 2 difference in CPU time. The edge imbalance is corrected by an enhanced geometrical partitioner described earlier. However, this did not improve the results since the number of fill-ins generated is not only influenced by the original nonzero elements but also by the manner in which the nodes are connected. Again, the best results are still obtained with perfect node load balance.

3.4.5.4 Effects of Ordering

As mentioned earlier, the reverse Cuthill-Mckee, minimum degree, and maximum degree algorithms [19] are implemented for ordering processors and grid nodes within each processor. No ordering is needed for the processors since the links across processors during preconditioning have been removed for parallel execution. From our experiences, the minimum degree grid node ordering gives the best convergence behavior for the ECL and LOCOS devices. Reverse Cuthill-Mckee ordering is observed to give the next best convergence behavior. The node with the minimum degree is used as the starting node for the reverse Cuthill-Mckee ordering. Both ordering schemes maintain a significant portion of coupling between nodes when fill-ins are allowed. Maintaining the coupling of nodes for rectangular grids has been illustrated by Tomacruz et al. [44] to give the best preconditioners. This also appears to be a good criteria for irregular grids.

3.4.6 Scalability

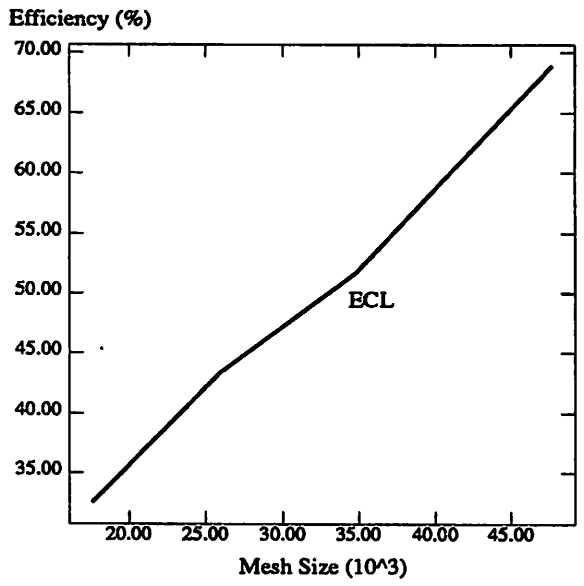

Figure 3.9: Scalability with problem size

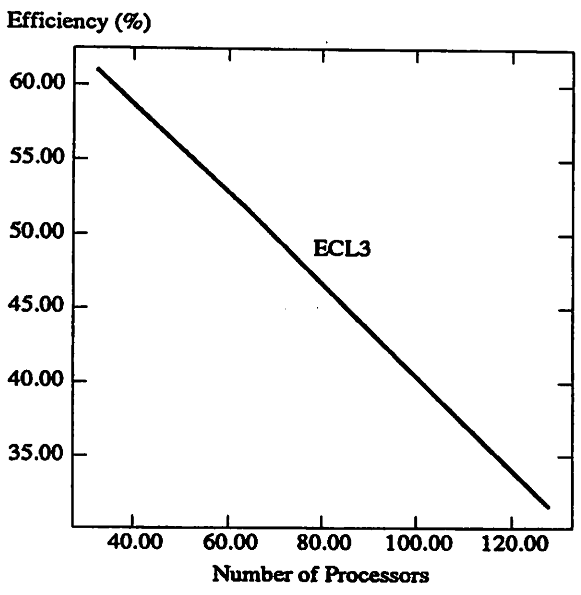

Figure 3.10: Scalability with machine size

Figure 3.9 shows the increase in efficiency as the problem size becomes larger. These results are obtained for a 64 node CM-5 using the ECL device. Sequential to parallel speed-up increases as the number of processors is increased. However, the efficiency of the algorithm is shown in Figure 3.10 to decrease as the number of processors increases. This is due to the fact that, for a static mesh size, the connectivity of the mesh structure is compromised further as the number of processors increases. This degrades the performance of the preconditioner which results in an increase of the total number of iterations required for convergence.

It should be pointed out that due to memory limitations of the sequential matrix generator, the largest mesh size solved is about 50,000 nodes. A major portion of three-dimensional device simulation applications are expected to require nodes in the order of a hundred thousand. Therefore, with larger parallel computers, we still expect a 50% computational efficiency with the simulation of larger device problems.

3.4.7 Discussion

Parallel computers are shown to be effective in doing irregular grid drift-diffusion device simulation. A 50% efficiency is obtained for the solution of large device matrices utilizing the iterative CGS linear solver with preconditioning. The best preconditioner observed uses incomplete LU decomposition with fill-ins. Preconditioning is parallelized by removing links between processors during ILU and FBS. Hence, communication calls are only necessary for dot product and matrix times vector operations. Fill-ins are generated using a magnitude threshold criteria that is adjusted to provide fill-ins comparable to the number of entries in the matrix being solved. The minimum degree node ordering is observed to give the best results. Due to the total parallelism of the preconditioner, no processor ordering is necessary.

Perfect node load balance is observed to be the most important partitioning parameter. The geometrical partitioner is the preferred partitioning algorithm since it can obtain perfect node load balance partitions an order of magnitude faster than the topographical and spectral partitioners. It can obtain this result and, at the same time, it tends to minimize average and maximum processor neighbors. Minimizing total edges cut, as obtained by the topographical partitioner, is not important since the CGS with preconditioning algorithm spends less than 10% of the total CPU time doing communication calls. The computational cost of the sequential geometrical partitioner is not significant since it is only done once, while a typical device simulation requires the solution of numerous time points or voltage points.

If a Cray-2 is used to run PILS, a 40X performance improvement over Sparc workstation performance is obtained [6]. Hence, it can be concluded that a 128 node CM-5 with no vector units will exceed Cray-2 performance for large irregular grid semiconductor drift-diffusion device simulations.

MFLOPS ratings for the solution of device matrices are not useful. The best algorithms (algorithm that minimizes wall clock time and has good convergence properties) are different for sequential and parallel implementations. The parallel algorithm will require more total floating point operations to converge since parallelization degrades the quality of the preconditioner.

The current implementation uses a sequential matrix generator which takes 10-30% of the total sequential CPU time [6]. This should be easy to parallelize since the communication requirement is the same as the matrix times vector operation. Also, the algorithm speed can be further increased with the use of vector units. Groups of nodes within each processor may be done in parallel with the vector units while using ILU with minimal fill-ins.

The CPU time required for a parallel direct solver is determined by the number of operations and the percentage that can be done in parallel. The ordering of the nodes determines the number of operations and the amount of parallel work that can be done. Evaluating these parameters is difficult for irregular grids. It has to predict the number of fill-ins and also areas of parallel processing given an ordering. An upper bound can be set by looking at direct dense matrix solvers. In computing the floating-point execution rate, use $2n^3/3 + 2n^2$ operations independent of the actual method used. Dongarra [30] reports 64 node CM-5 performance of 3.8 Gflop/s for LINPACK. Hence, a 64 node CM-5 direct dense matrix solver would take more than 48 hours to solve a 100,000 node matrix. It is still not clear how well sparse direct solvers can be implemented on a parallel machine. Algorithms of $O(n^x)$ where x is less than 2 may be possible for sparse matrices with a specific structure. Whether these algorithms may be used with good parallel efficiency for the irregular problems defined in this chapter still needs to be determined.

3.5 Extensions

3.5.1 Active Messages and Workstation Clusters

Martin [31] or von Eicken et al. [32] network report communication latencies comparable to MPPs. It is not clear if such special network set-ups will be available in future workstation clusters. These future clusters will be characterized by the use of off-the-shelf components that address a wide range of task requirements. Hence, implementation of the parallel algorithms on workstation clusters would only be confined to current available technology.

Tables 1.5 shows message-passing times in the order of milliseconds for an unloaded workstation and an unloaded network. Table 1.6 illustrates broadcast and barrier synchronization times in the order of tens of milliseconds for a 32 node cluster. These communication latencies may be hidden through the use of active messages. The efficient use of active messages may determine the usefulness of workstation clusters for parallel TCAD simulations.

The presented algorithms revolve around the efficient solution of the preconditioned CGS algorithm. Active messages will not significantly improve the performance of an MPP implementation since only 10% of the current CPU time is spent on communication. However, active messages may be able to hide the latency inherent in workstation clusters. The first criterion to check is the time for each iteration. For the devices presented in this chapter, each iteration CPU Time is in the order of the hundreds of milliseconds which would imply the possible applicability of active messages. The focus would then be on dot products and matrix times vector operations since they require communication calls.

Active messages can be applied to dot products by making each processor send their local dot products to a host processor. The host processor can then add up the partial dot products and send it back to the processors. It is currently not clear how much computation can be done while the processors are waiting for the dot product. By rearranging some operations like moving the update of x to location after the first dot product in the inner loop and by having large enough data set within each processor, physical latency due to dot products may be hidden by active messages.

Active messages can also be efficiently used in the matrix times vector routine since each processor can first send all the data needed by its neighbors and then work on its own data while waiting for the data it needs from its neighbors. Again, the physical latency can be hidden by having large enough data within each processor.

Hence, by working on large data within each processor and by careful arrangement of linear algebra operations, it may be possible to use active messages to efficiently implement CGS routines on workstation clusters. However, Lewis and van de Geijn [33] report that the use of active messages complicates the implementation of matrix times vector routines for an Intel iPSC/860. The running time of their algorithm is difficult to estimate accurately because each node may be computing and have numerous messages in transit at any time. A more crucial issue is the rapidly increasing number of messages which may prevent the algorithm from scaling to a very large number of nodes. It should be pointed out that high efficiency may not be necessary to justify the use of workstation clusters since workstations for sequential computing should already justify the acquisition and operating cost.

3.5.2 Explicit Methods

Tai et al. [36], Kurata et al. [35], and Pleumeekers et al. [34] present explicit drift-diffusion device simulators. Explicit methods do not require the solution of a linear system of equations. Hence, they have small memory requirements and are simple to implement both in sequential and parallel form. Tai et al. [36] presents a parallel explicit 2-D device simulator that has a speed-up of 8 for an SOI application due to parallelism in the time domain. However, no comparisons are made between implicit and explicit methods. The 2-D device was mapped into an 16K CM-2 machine by making multiple copies and mapping each grid point to a processor. Each device represents a time point and the speed-up was then calculated based on the performance of a single device simulation. Kurata et al. [35] uses an explicit algorithm on 2-D bipolar transistor examples to obtain results that take 5 to 10 times more CPU times compared to the implicit device simulator TONADDE2. Pleumeekers et al. [34] performed 3-D explicit simulations on diodes, BJTs, and FETs. In all cases, the solutions were obtained with a factor of 10 to 50 degradation in CPU time compared to implicit simulations. For these results and the previous observation of 50% parallel processing efficiency for implicit methods, it can be concluded that even with 100% parallel efficiency for current explicit methods, the most efficient parallel device simulator will still be based on implicit methods.

3.5.3 Adaptive Grids

Three-dimensional adaptive semiconductor device simulation have been presented by Coughran et al. [38] and Burgler et al. [39]. Adaptive grid simulators modify mesh structures during simulation to compensate for unpredictable and changing fields and currents in order to achieve accuracy and efficiency. Adaptive grids are particularly useful for time-dependent simulations where the active region of the device changes in time.

Coughran et al. [38] illustrates a typical outline of a grid generation procedure used by adaptive grid simulators. The process simulator supplies the topology and the doping profile to the initial grid generator. The adaptive refinement uses the doping profile, current mesh, and current solution as a basis for creating the new grid. Using this high-level procedure, the applicability of parallel processors to 3-D adaptive device simulation is examined.

Figure 3.11 illustrates a proposed adaptive grid simulator. It uses the sequential geometrical partitioner and the parallel static grid simulator presented earlier. Several new issues need to be addressed - initial grid generation, error indicators, and parallel adaptive refinement.

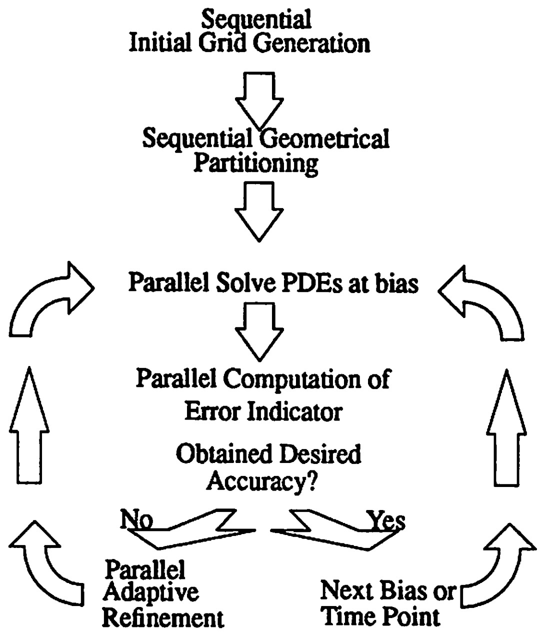

Figure 3.11: Proposed Parallel Adaptive Grid Simulator

The goal is to minimize the number of adaptive refinement iterations needed to obtain the desired accuracy. The initial grid should also not be overly fine since this will degrade the efficiency of the program. A good initial grid generator is OMEGA [11]. To generate a mesh structure with about 100,000 nodes for a CMOS inverter would take 600 sec on a SUN-SPARC1 [10]. Using parallel processors comparable in performance to traditional vector supercomputers (128 node CM-5 = Cray-2 from previous section), an Omega initial grid generator would require negligible CPU time compared to the 5 hrs required for the actual device simulation [6]. Based on extrapolation of the data in Table 3.1, partitioning can be done in less than 300 sec which again can be neglected in terms of total CPU time.

Local error estimators based on the formalism proposed by Bank and Weiser [41] has been found to give excellent results [38], [39], [40]. An edge is refined based on an error computation that only requires data from the two control volumes the edge connects. The scheme has been shown to be successful in simulating bipolar transistors, MOS devices, and CCD [38] [39]. Due to the locality, error evaluations can be done in parallel.

From the previous paragraph, there exists for each element an indicator telling across which axes of the element needs further refinement. Refinement rules presented by Hitschfeld et al. [11] can then be applied. This process may be done on a sequential or parallel manner. A sequential methods avoids the problem of implementing a parallel Omega algorithm. However, when grids points are added or deleted for the next simulation, clever allocation routines are needed to be developed. These routines should be knowledgeable of the current partitioning of nodes, be able to maintain load balance by moving grid points from one processor to its neighbors, and be able to easily modify the communication scheduling algorithm. The second option of implementing a parallel mesh generator would also need these clever allocation routines and at the same time be able to efficiently represent a parallel mesh data structure. One difficulty lies on grid points at the subdomain boundaries which are made denser or coarser. This may require communication with other processors.

3.5.4 Hydrodynamic Models

Drift-diffusion device simulators ignore the spatial variation of the average carrier energy and assume that the mobility and diffusion parameters are uniquely specified by the local electric field. These limitations can be removed by adding the energy balance equation which will produce a set of equations called the hydrodynamic equations since the system is similar to that which describes the flow of fluids. The energy balance equations are derived from the second-order moments of the Boltzmann Transport Equation (BTE) [43,44].

$$\frac{\partial W_C}{\partial t} = -\nabla \bullet \vec{F}_{W_C} + \vec{J}_C \bullet E + \left(\frac{\partial W_C}{\partial t}\right)_{COLL} \tag{3.6}$$

where W, F, J, C, and E are the energy densities, energy flows, current densities, carrier type (n or p), and electric field, respectively. The energy flow is defined as:

$$\vec{F}_{w_c} = -\chi_c \nabla T_c - \left(\frac{5}{2} - s\right) \frac{k_B T_c}{q} \vec{J}_c \tag{3.7}$$

where T_c represents the temperature, χ_c the related thermal conductivity, and s is a weakly-varying correction, related with the dominant scattering mechanism. The hydrodynamic model extends the drift-diffusion model through three modifications on the particle flux density equation. The first term is the diffusion current for which a generalized Einstein relation between mobility and diffusivity can be recognized. Second, the E term is modified to include the thermoelectric contribution which arises from the gradients in the carrier temperature. And finally, a third term is added to represent contribution from the drift energy.

$$\vec{J}_C = \pm q\mu_C \left[\frac{k_B T_C}{q} \nabla C + C \nabla \left(\frac{k_B T_C}{q} \pm \psi\right)\right] + \frac{\mu_C m_n}{q^2} \vec{J}_C \nabla \bullet \frac{\vec{J}_C}{C} \tag{3.8}$$

The average energy density of the carrier subsystems can be expressed as follows:

$$W_n = \frac{3}{2} k_B T_C C + \frac{1}{2} m_C v_C^2 C \tag{3.9}$$

where m is the carrier effective mass and v the drift-velocity. In the usual energy transport model, the second term is neglected. In the hydrodynamic model (HD), this assumption is not made, and the drift energy is included in the kinetic energy tensor.

Energy balance and HD device simulators do not have the same region of validity as Monte Carlo device simulators. Monte Carlo calculates the full distribution function without assumptions of its form. Any moment-based approach (energy balance or full hydrodynamic) only obtains the first few moments of the distribution function. A good example is page 105 of Lundstrom [45]. This shows the distribution in the base of a heterojunction bipolar transistor (HBT). In a thin base HBT, most of the current is carried in the "ballistic peak". Using only the energy balance equation will not model this properly. Hydrodynamic simulation may be able to model it. However, anything sensitive to the "thermalized" part of the distribution would probably not be simulated accurately by HD. The base recombination current might be a parameter that is strongly dependent on the thermalized part which would be neglected by HD.

There are several versions of HD simulators. Pierantoni et al. [44] features an accurate description of the energy exchange among electrons, holes, and lattice, and is therefore suitable for self-consistently simulating thermal effects and non-stationary phenomena, as well as their possible interactions. Ahn et al. [46] presents a hydrodynamic model that accurately computes the high energy tail electrons which contributes to the impact ionization and the injection of current into the gate oxide. Ramaswamy et al. [47] makes a survey of some hydrodynamic models and presents a scheme to determine the accuracy of the models.

Gardner et al. [48] demonstrated that block successive under relaxation converges for a 1-D submicrometer semiconductor device using the hydrodynamic model. They obtained a parallel speed-up of approximately 2.5 on 10 processors using a chaotic relaxation and the preconditioned conjugate gradient method for the parallel diagonal block solver.

In implementing the hydrodynamic simulator, the presented framework and algorithms may again be used. There are now six unknowns that can be solved together or in some decoupled scheme. Implementing a good multigrid initial guess routine along with a good preconditioner may again determine the usefulness of a MP HD simulator. Other issues to be resolved are the mesh structure type and the grid generation algorithm to be used.

Sample matrices are obtained from Pierantoni et al. [44]. Unfortunately, the matrices are not arranged in accordance to the grid point they belong to. Also, no completely coupled solution of the thermal-hydrodynamic model is available. The solution is divided into two blocks - $[\psi, n, p]$ and $[t_n, t_p, t_l]$. One sample matrix came from a BJT device. The total number of equations is 52908 - the first 17637 are the Poisson equations, the second 17637 block are for the electron equations and the last block are for the hole equations. This particular matrix is solved by a sequential CGS solver in 327 iterations (253 seconds in a 365-IBM RISC 6000). Another sample matrix came from a MOS simulation. It had 20000 equations and it took the same sequential machine 33 seconds to solve. Using the parallel solver, presented in this chapter, no speed-up is obtained due to the small sizes of the matrices and the inefficient partitioning of the nodes. However, the conditioning of matrices is comparable to regular drift-diffusion since the parallel solver takes less than 100 iterations to solve the matrices. Hence, it is expected that the parallel solver will show significant speed-up for larger problems.

An important issue is when does MP solution of BTE moments become more computationally expensive compared to MP Monte Carlo solution. Sequential Monte Carlo simulations are an order of magnitude slower than sequential thermal-hydrodynamic simulations. For sequential Monte Carlo simulation, Brisset et al. [49] reports that contrary to the case of 2D modeling, the CPU time used for solving Poisson's equation becomes non-negligible. The total CPU time is strongly dependent on the number of simulated particles, the number of meshes, and the time step. For a 20 x 20 x 40 mesh, 12500 particles and a 1 *fs* time step, the simulation of 10 *ps* on SUN Sparc 10/20 took 28.5 hours. 16.5% of the CPU time was spent on calculating the motion of the particles while 77% of the CPU time was spent solving Poisson's equation.

Section 1.3.2 presents four publications on parallel Monte Carlo semiconductor device simulation. Ranawake et al. [50] reports a 70% efficiency for 256 x 16 x 128 mesh structure. Even by assuming a 100%

efficiency for parallel Monte Carlo algorithms and a 50% efficient for parallel hydrodynamic simulators, hydrodynamic simulators are still expected to be an order of magnitude faster.

Another factor that may influence a user's choice of models is ease of use in obtaining accurate results. Ease of use is a combination of several factors. First, the user needs to discretize the problem accurately for meaningful results. For hydrodynamic simulation, the user may specify simple rectangular grids or complicated irregular grids to minimize the number of grid points. For 3-D Monte Carlo, rectangular grids are usually used since this allows easy book keeping of particles for each subdomain. Second, the user needs to setup the simulation parameters. Both methods require careful choice of simulation time steps. For hydrodynamic models, other parameters such initial bias conditions and the collision terms which are calculated by relaxation-time approximation need to provided. In many cases, Monte Carlo simulations are needed to give the right hydrodynamic parameters. For Monte Carlo, the user would need to specify the positions and velocities of mobile carriers. There are also several physical parameters that need to be set depending upon the problem characteristics such as the type of material used. Third, the convergence behavior of the solution methods is another important consideration to the user. The block newton iteration with an iterative linear solve inner loop typically used for hydrodynamic simulation may not converge. Monte Carlo simulators usually produce simulation results after each run. However, its accuracy is dependent upon the parameter setting. Finally, the user must have confidence on his models. Monte Carlo simulators definitely have more accurate models. Hence, several papers have been published that attempts to combine drift-diffusion and monte-carlo to retain the computational efficiency of the pde-based method as well as the accuracy of the Monte Carlo technique. Kosina and Selberherr [51] describe the basic ideas of the coupled technique.

3.5.5 Noise Simulation for Nonlinear Dynamic Circuits

The iterative solver developed in this chapter is also used for circuit noise simulation. Noise represents a lower limit to the size of electrical signals that can be amplified without significant deterioration in signal quality. It also represents an upper limit for the useful amplifier gain. Demir et al. [37] presents a time-domain non-Monte Carlo noise simulation for nonlinear dynamic circuits with arbitrary excitations.

The noise simulation method was implemented inside the circuit simulator SPICE3. Time domain simulation is done along with the transient simulation in SPICE3. The transient simulation in SPICE3 solves the MNA equations without noises for the circuit. The circuit variables consist of node voltages and branch currents for some elements. The circuit equations consist of the node equations and branch equations of the elements for which branch currents are included in the circuit variables vector.

The noise simulation is done concurrently with the transient simulation. First, the stochastic differential equation for noise is derived from MNA formulation of the nonlinear circuit equations. Second, the stochastic differential equation for noise is transformed in state-equation form. Finally, the stochastic differential equation for noise is solved.

$$\dot{K}^1(t) = E(t)K^1(t) + K^1(t)E(t)^T + F(t)F(t)^T \tag{3.10}$$

where $K^1(t)$ represents the noise covariance matrix of circuit variables as a function of time, $E(t)$ represents a matrix which is a function of the derivatives of the MNA equations, and $F(t)$ is the MNA equation for the circuit. Equation 3.10 is a system of *m(m+1)/2* linear differential equations where *m* is close to the number of nodes. The Backward Euler scheme is used to solve the equation which transforms the problem into the solution of a sparse linear system of equations. The system of linear equations would be tridiagonal if the second term of Equation 3.10 was not included. This second term complicates the solution process due to the need of general purpose sparse matrix solvers.

Direct methods are first investigated for the solution of the linear system. A BJT active mixer with 65 nodes is used as an example. The noise simulation requires the solution of 2145 linear equations at each of the 250 time points. This took approximately 17 hours on a DECstation 5900/260. 95% of the CPU time is used for the solution of large sparse matrices using a direct solver. Using a sequential iterative solver, the total simulation time was reduced to 0.5 hours. CGS with no preconditioning implemented in PILS [12] was used as the iterative linear solver. Parallel solution of this problem took 20x250 seconds.

Another example involved the noise simulation of a 16 component delay line in which each component had 13 transistors. The matrix generated by this example has 29403 rows. Due to the size of the problem, direct solvers are not feasible due to memory and speed limitations of current computing platforms. PILS running on a MIPS R4400/60MHz processor took 1950 seconds to solve the D.C. state noise matrix. The parallel solver took 330 seconds to calculate the same solution. Since MIPS R4400/60MHz is more 2 times faster than a Sparc II (CM-5 processors), an efficiency of more 20% is observed. Larger problems are expected to give better efficiency results.

3.6 Summary

The use of parallel processors for the solution of drift-diffusion semiconductor device equations using an irregular grid discretization has been developed and published [52]. Preconditioning, partitioning, and communication scheduling algorithms are developed to implement an efficient and robust iterative linear solver with preconditioning. The parallel program is executed on a 64 node CM-5 and is compared with PILS running on a single processor. We observe an efficiency increase in obtaining parallel speed-ups as the problem size increases. A 60% efficiency for CGS with no preconditioning is obtained for large problems. Using CGS with processor ILU and magnitude threshold fill-in preconditioning for the CM-5 and CGS with ILU for PILS, a 50% efficiency for the solution of the large matrices is attained. Perfect node load balance is observed to be the most important partitioning parameter.

In trying to improve the simulator, explicit methods are shown not be as efficient compared to implicit methods for the drift-diffusion equations. An adaptive grid algorithm is described along with the difficulties in its implementation. The proposed irregular grid algorithm is also used in solving the hydrodynamic and circuit noise equations. It is observed that the hydrodynamic and drift-diffusion matrices have comparable conditioning. Also, for large hydrodynamic and circuit noise problems, good parallel efficiency are expected to be observed.

References

[1] D. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, and A. Sangiovanni-Vincentelli, "A Massively Parallel Algorithm for Three-Dimensional Device Simulation," *IEEE Trans. on CAD*, Vol. 10, pp. 1201-1209, Sept. 1991.

[2] E. Tomacruz, J. Sanghavi, and A. Sangiovanni-Vincentelli, "Algorithms for Drift-Diffusion Device Simulation Using Massively Parallel Processors," *IEICE Trans. on Electronics*, Vol. E77-C, pp. 248-254, Feb. 1994.

[3] K. Wu, G. Chin, and R. Dutton, "A STRIDE Towards Practical 3-D Device Simulation - Numerical and Visualization Considerations," *IEEE Trans. on CAD*, Vol. 10, pp. 1132-1140, Sept. 1991.

[4] R. Dutton, "Algorithms and TCAD Software using Parallel Computation," *VPAD Proceedings*, pp. 10-12, 1993.

[5] W. Coughran, M. Pinto, and R. Smith, "Adaptive Grid Generation for VLSI Device Simulation," *IEEE Trans. on CAD*, Vol. 10, pp. 1259-1275, Oct. 1991.

[6] G. Heiser, C. Pommerell, J. Weis, and W. Fichtner, "Three-Dimensional Numerical Semiconductor Device Simulation: Algorithms, Architectures, Results," *IEEE Trans. on CAD*, Vol. 10, pp. 1218--1230, Oct. 1991.

[7] M. Noell, S. Poon, M. Orlowski, and G. Heiser, "Study of 3-D Effects in Box Isolation Technologies," *SISDEP Proceedings*, pp. 331-340, 1991.

[8] L. Bosisio, F. Forti, and E. Tomacruz, "Measurement and Tridimensional Simulation of Silicon Pixel Detector Capacitance," *IEEE Nuclear Science Symposium*, Vol. 1, pp. 338-342, 1993.

[9] J. Palmer, and G. Steele Jr., "Connection Machine Model CM-5 System Overview," *IEEE 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 474-483, 1992.

[10] P. Conti, N. Hitschfeld, and W. Fichtner, "Omega - An Octree-Based Mixed Element Grid Allocator for the Simulation of Complex 3-D Device Structures," *IEEE Trans. on CAD*, Vol. 10, pp. 1231-1241, 1991.

[11] N. Hitschfeld, P. Conti, and W. Fichtner, "Mixed Element Trees: A Generalization of Modified Octrees for the Generation of Meshes for the Simulation of Complex 3-D Semiconductor Device Structures," *IEEE Trans. on CAD*, Vol. 12, pp. 1714-1725, Nov. 1993.

[12] C. Pommerell and W. Fichtner, "PILS: An Iterative Linear Solver Package for Ill-Conditioned Systems," *Supercomputing'91*, pp. 588-599, 1991.

[13] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*, Springer-Verlag, Wien, Austria, 1984.

[14] N. Hitschfeld, K. Kells, P. Conti, *Omega 3.0 User's Guide*, Integrated Systems Laboratory, ETH-Zentrum, Zurich, 1991.

[15] D. Scharfetter and H. Gummel, "Large-Signal Analysis of a Silicon Read Diode Oscillator," *IEEE Trans. Electron Devices*, Vol. 16, pp. 64-77, 1969.

[16] R. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[17] P. Sonneveld, "CGS, A Fast Lanczos-type Solver for Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Comp.*, Vol. 10, pp. 36-52, Jan. 1989.

[18] H. Van der Vorst, "Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, Vol. 13, pp. 631-644, 1992.

[19] I. Duff and G. Meurant, "The Effect of Ordering on Preconditioned Conjugate Gradients," *BIT*, pp. 635-657, 1989.

[20] M. Berger and S. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors," *IEEE Trans. on Computers*, Vol. C-36, pp. 570-580, May 1987.

[21] C. Fiduccia and R. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Design Automation Conference Proceedings*, Vol. 19, pp. 175-181, 1982.

[22] W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, pp. 291-307, Feb. 1970.

[23] H. Simon, "Partitioning of Unstructured Problems for Parallel Processing," *Computing Systems in Engineering*, Vol. 2, pp. 135-148, 1991.

[24] M. Fiedler, "A Property of Eigenvectors of Nonnegative Symmetric Matrices and Its Application to Graph Theory," Czechoslovak Mathematics Journal, Vol. 25, pp. 619-633, 1975.

[25] Z. Bozkus, S. Ranka, and G. Fox, "Benchmarking the CM-5 multicomputer," *IEEE 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 100-107, 1992.

[26] *CMMD Reference Manual*, Thinking Machines Corporation, Cambridge Massachusetts, 1992.

[27] N. Hitschfeld, P. Conti, and W. Fichtner, "Grid Generation for 3-D Nonplanar Semiconductor Device Structures," *SISDEP Proceedings*, Vol. 4, pp. 165-169, 1991.

[28] Z. Zhao, Q. Zhang, G. Tan, and J. Xu, "A New Preconditioner for CGS Iteration in Solving Large Sparse Nonsymmetric Linear Equations in Semiconductor Device Simulation," *IEEE Trans. on CAD*, Vol. 10, pp. 1432-1440, Nov. 1991.

[29] K. Kundert, "Sparse Matrix Techniques and Their Application to Circuit Simulation," *Circuit Analysis, Simulation, and Design*, A. Ruehli ed., North-Holland, New York, NY, 1986.

[30] J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software," *Oak Ridge National Laboratory Tech. Rep.*, 1993.

[31] R. Martin, "HPAM: An Active Message Layer for a Network of HP Workstations," *IEEE Hot Interconnects II*, Aug. 1994.

[32] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. of the 19th ISCA*, pp. 256-266, May 1992.

[33] J. Lewis and R. van de Geijn, "Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithms," *Supercomputing '93*, pp. 484-492, 1993.

[34] J. Pleumeekers, C. Simon, S. Mottet, "An Explicit Method for Solving the Semiconductor Equations," *Nasecode X Proceedings*, pp. 99-100, 1994.

[35] M. Kurata, S. Nakamura, "An Explicit Method of Numerical Integration for the Complete Set of Semiconductor Device Equations," *IEEE Trans. on CAD*, Vol. 11, pp. 1013-1013, Aug. 1992.

[36] G. Tai, C. Korman, and I Mayergoyz, "A Parallel-In-Time Method for the Transient Simulation of SOI Devices with Drain Current Overshoots," *IEEE Trans. on CAD*, Vol. 13, pp. 1035-1044, Aug. 1994.

[37] A. Demir, E. Liu, A. Sangiovanni-Vincentelli, "Time-Domain non-Monte Carlo Noise Simulation for Nonlinear Dynamic Circuits with Arbitrary Excitations," *Proceedings of ICCAD*, 1994.

[38] Coughran, W., Pinto, M., and Smith, R., "Adaptive Grid Generation for VLSI Device Simulation,"

IEEE Trans. on CAD, Vol. 10, pp. 1259-1275, Oct. 1991.

[39] J. Ashby, C. Fitzsimons, R. Fowler, and C. Greenough, P. Mole, and W. Schilders, "Three-Dimensional Adaptive Semiconductor Device Simulation," *SISDEP Proceedings*, pp. 467-476, 1991.

[40] J. Burgler, W. Coughran Jr., and W. Fichtner, "An Adaptive Grid Refinement Strategy for the Drift-Diffusion Equations," *IEEE Trans. on CAD*, Vol. 10, pp. 1251-1258, Oct. 1991.

[41] R. Bank and A. Weiser, "Some a posteriori error estimators for elliptic partial differential equations," Math. Comp., Vol. 44, No. 170, pp. 283-301, Apr. 1985.

[42] M. Yerry and M. Shephard, "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique," Int. J. Numer. Methods Eng., Vol. 20, pp. 1965-1990, 1984.

[43] M. Lundstrom, *Fundamentals of Carrier Transport*, Addison-Wesley, Reading, Massachusetts, 1990.

[44] A. Pierantoni, P. Ciampolini, A. Liuzzo, and G. Baccarani, "A Unified Model for the Simulation of Small-Geometry Devices," *IEICE Trans. on Electronics*, Vol. E77-C, pp. 139-147, Feb. 1994.

[45] M. Lundstrom, *Fundamentals of Carrier Transport*, G. NeuDeck, and R. Pierret, eds., Addison-Wesley, Reading, Massachusetts, 1990.

[46] J. Ahn, Y. Park, and H. Min, "A New Hydrodynamic Model for High Energy Tail Electrons," VPAD Proceedings, pp. 28-29, 1993.

[47] S. Ramaswamy and T. Tang, "Comparison of Semiconductor Transport Models Using a Monte Carlo Consistency Test," *IEEE Trans. on Elec. Dev.*, Vol. 41, pp. 76-83, Jan. 1994.

[48] C. Gardner, P. Lanzkron, and D. Rose, "A Parallel Block Iterative Method for the Hydrodynamic Device Model," *IEEE Trans. on CAD*, Vol. 10, pp. 1187-1192, Sept. 1991.

[49] C. Brisset, P. Dollfus, N. Chemarin, R. Castagne, and P. Hesto, "Three-Dimensional Monte Carlo Simulation of Submicronic Devices," *Simulation of Semiconductor Devices and Processes*, S. Selberherr, H. Stippel, and E. Strasser, eds., Vol. 5, pp. 189-192, Sept. 1993.

[50] U. Ranawake, C. Huster, P. Lenders, and S. Goodnick, "PMC-3D: A Parallel Three-Dimensional Monte Carlo Semiconductor Device Simulator," *IEEE Trans. on CAD*, Vol. 13, pp. 712-724, 1994.

[51] H. Kosina and S. Selberherr, "A Hybrid Device Simulator that Combines Monte Carlo and Drift-Diffusion Analysis", *IEEE Trans. on CAD*, Vol. 13, pp. 201-210, Feb. 1994

[52] E. Tomacruz, J. Sanghavi, A. Sangiovanni-Vincentelli, "A Parallel Iterative Linear Solver for Solving Irregular Grid Semiconductor Device Matrices," *Supercomputing '94*, pp. 24-33, 1994.

CHAPTER 4

Reaction Kinetics and Diffusion Simulation in Chemically Amplified Resist

4.1 Background

Various lithographic strategies that have a primary difference in the type of radiation used to expose the resist have been developed over the past 30 years. All these strategies are limited by resist sensitivity which must be commensurate with the exposure parameters of the lithography equipment for efficient production. Factors such as less efficient radiation sources, photon dependency chemical transformations, or increase in wafer size from four inches to eight inches and beyond may necessitate the need for increased resist sensitivity. Chemically amplified resist systems which may be two orders of magnitude more sensitive [2] are being developed to maintain and improve productivity in terms of wafer exposures per hour. Chemically amplified systems produce an acid through a photolytic reaction in the initial exposure. The acid then catalyzes a second chemical reaction during post-exposure bake. The extent of the catalytic reaction determines the dissolution rate during development. Figure 4.1 demonstrates a general process flow for a chemical amplification resist consisting of the exposure, a critical post-exposure bake, and development steps.

Chemically amplified resist systems have both nonlinear chemical reaction kinetics [12] and simultaneous concentration dependent diffusion. Experimental results such as the lost of standing wave features (Figure 4.5) and T-topping suggest acid diffusion. Linear linewidth change experimental results presented by Zuniga et al. [32] and illustrated by Figure 4.11 show evidence that the diffusion is concentration dependent. The level of complexity in modeling these reaction and diffusion effects is similar to that involved in modeling impurity concentration and point defect dependent diffusion in silicon. New mathematical methods in conjunction with experimental results are needed to be developed to verify reaction parameters. The use of high post-bake temperatures is preferred as resist sensitivity improves. However, it also results in increased diffusion and its effects on resolution must be examined in three dimensions. The nonlinearity of the reactions and the dependence of the diffusion on the local concentration can lead to improved performance. Understanding and balancing these mechanisms is the key goal in designing production worthy resists.

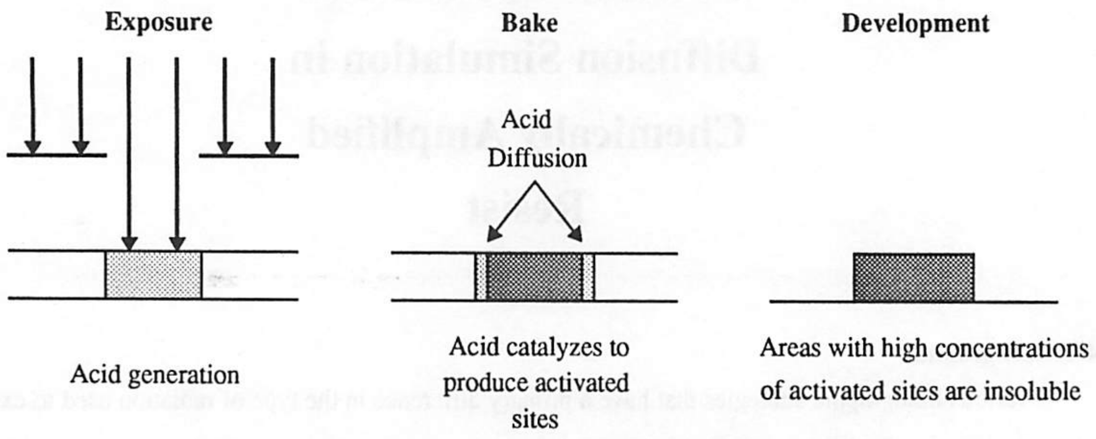
Exposure
Bake
Development
Acid Diffusion
Acid generation
Acid catalyzes to produce activated sites
Areas with high concentrations of activated sites are insoluble

Figure 4.1: Process Flow for a Chemical Amplification Resist

This chapter models the 3-D movement and reaction of species in the post-exposure bake of chemically amplified resist systems. Both weak and strong dependencies of diffusion on species concentration are considered by investigating several diffusion models. Combinations of nonlinear reaction kinetics and concentration dependent diffusion scenarios are being considered for both acid-hardening (negative) resist and deprotection reaction (positive) resist. Specifically, we focus our studies on the Shipley SNR-248, I.B.M. Apex-E, and I.B.M. Apex-M chemically amplified resists. Due to the predicted large computational requirements of 3-D simulations, the target architectures of algorithms developed in this chapter are parallel processors.

This chapter is organized as follows: Section 4.2 describes the equations being solved. Computational steps to solve the problem implicitly and parallel processor programming consideration are illustrated in Section 4.3. Section 4.4 explains the discretization method and the matrix assembly equations are described in Section 4.5. The diffusion models are examined in section 4.6. Section 4.7 describes possible applications and justifies the use of parallel processors. Section 4.8 presents an adaptive grid method that may be used to accelerate the simulations. Section 4.9 explores possible extensions and applications of the algorithms presented.

4.2 Computational Models

4.2.1 Modeling the Exposure

The SPLAT program [19] is used to determine the intensity distribution as a function of the user specified mask. The program is capable of simulating the effects of lens aberrations, apodization, spatial filtering, focus, and magnification effects for high NA, and modified illumination. The intensity distribution is a result of the coherent and incoherent superposition of a large number of multiply reflected waves within the thin-film structure. It uses the Hopkins theory of partially coherent imaging to simulate projection-printing with partial coherence. SPLAT extends Hopkins theory to simulate non-vertical propagation effects in thin-film interference. Such effects have been shown to be important in photolithography when the Rayleigh depth of focus $R = \lambda/2(NA)^2$ is comparable to or less than the thickness h of the photoresist divided by the real part n of its refractive index. The mask is specified as a set of rectangles and triangles whose size, position, and relative transmittance are specified by the user.

To generate a 3-D distribution of absorbed energy, the SPLAT program has an independent calculation for each layer in the third dimension. This may be inefficient since symmetry calculations need only be done for one z-layer and it may be possible to compute intensity at a smaller number of z-layers and to use interpolation for the layers between. Helmsen et al. [28] reports that for a 50x50x93 mesh structure, an I.B.M. RISC-6000 took 6.8 hours to run Splat. By running this program on a parallel machine or a group of workstations, each layer can be done concurrently. Since there is no need for communication, Splat can be executed with close to perfect parallel speed-up efficiency.

Dill's ABC model [24] can be used to simulate the generation of acid during the exposure through a simple photolytic reaction. Dill's equations are summarized below.

$$\frac{\partial}{\partial z} I(z,t) = -(AM(z,t) + B) I(z,t) \qquad (4.1)$$

$$\frac{\partial}{\partial t} M(z,t) = -CM(z,t) I(z,t) \qquad (4.2)$$

where I and M are the intensity distribution and the normalized concentration of photoactive compound respectively. A, B, and C are Dill's fitting parameters.

4.2.2 Modeling the Post-Exposure Bake

In chemically amplified resists, one photogenerated molecule drives several reaction cycles during a post-exposure bake. Ferguson et al. [12] describes the onium salt bake model which is used in this research project. Although much work has been done, many of the mechanisms such as diffusion which determine resist performance are still not well understood. Hence, we develop a general model for two interacting and one diffusing species which is summarized by the equations below.

$$\frac{\partial C_1}{\partial t} = k_1 (1 - C_1) C_2^m \tag{4.3}$$

$$\frac{\partial C_2}{\partial t} = \nabla \bullet (D_2 \nabla C_2) + k_2 C_2 \tag{4.4}$$

$$\frac{\partial}{\partial z} C_2 (x, y, z) = 0|_{Boundary} \tag{4.5}$$

$$C_1 (x, y, z) = 0|_{t=0} \tag{4.6}$$

$$C_2 (x, y, z) = C_S|_{t=0} \tag{4.7}$$

where C_1 is the concentration of activated sites, C_2 is the concentration of acid, k_1 is the reaction rate coefficient, k_2 is the rate coefficient for the acid loss reaction, and m is a fitting parameter (m>1). D_2 is the diffusion coefficient which may be dependent on C_1. Equation 4.3 describes the removal of the t-BOC protecting groups. Equation 4.4 is the mathematical theory of diffusion in isotropic substances which is based on the hypothesis that the rate of transfer of a diffusing substance through the unit area of a section is proportional to the concentration gradient measured normal to the section. The second term of Equation 4.4 describes acid loss. Equation 4.5 specifies that no net flow of acid occurs across the simulation boundaries. Equation 4.7 sets the initial value of the acid by using Dill's equations and the exposure data generated by SPLAT [19].

The diffusion coefficient D_2 is not well understood and might be dependent on 1) the acid concentration itself (constant), (2) the presence of deprotection sites which provide additional stepping stones (linear model), or (3) the increase in free volume with the deprotection reaction which creates a very rapid increase in diffusion pathways (exponential model). The free volume theory is based on the assumption that a diffusing molecule can only move from one place to another when the local free volume around that molecule exceed a certain critical value [31]. Relatively small changes in free volume can lead to a large change in the diffusion coefficient [14] which can be modeled using an exponential equation. These three possibilities are illustrated by Equations 4.8 - 4.10. The second and third models are type II diffusion models which are defined to have concentration dependent coefficients. To explore the possibility of these various classes of acid concentration and material state dependent diffusion, a general purpose 3-D reaction-diffusion simulator is proposed. For each of the three classes of increasingly higher nonlinearity an appropriate algorithm is chosen.

$$D_2 = \alpha \tag{4.8}$$

$$D_2 = \gamma + \beta C_1 \tag{4.9}$$

$$D_2 = \lambda exp(\omega C_1) \tag{4.10}$$

where α, γ, β, λ, and ω are constant parameters.

4.3 Computational Steps

Rectangular grids with nonuniformly spaced lines are used for the discretization. The second order trapezoidal method is used for numerical integration and the nonlinear equations are solved using the Newton-Raphson method. The Conjugate Gradient Squared (CGS) iterative algorithm described by Section 2.4.1 is used to solve the unsymmetric sparse matrix. Incomplete LU decomposition described by Section 2.4.2 with the red-black ordering defined by Section 2.4.3.1 is used as a preconditioner for the linear solver. These computational steps can be summarized by Algorithm 4.1.

The CM-2 is used as the target machine for the algorithms. Grid point partitioning described by Section 2.3.3.1 is used for parallelization. With 256k bits of memory for each processor, a virtual processor ratio of 64 is obtained. Hence, an 8k CM-2 with 256k bits of memory for each processor would allow a user to simulate a 512k mesh structure.

Algorithm 4.1: Time-Dependent Computational Steps

```
problem read-in and setup
time integration loop
  Newton-Raphson loop
      evaluate the equations for the Jacobian and
          right-hand side of the Newton iteration
      solve the associated linear system
post-processing of results
```

4.4 Discretization

Ferguson [27] uses a second-order Taylor series approximation for the spatial derivatives for constant diffusion coefficients. For concentration dependent diffusion coefficient scheme, a first order Taylor series approximation is used.

Since the exponential diffusion coefficient is a highly non-linear function of the variables, the standard difference discretization is not suitable for the task unless the grid spacings are made very small. To attain a more stable discretization, a technique proposed by Scharfetter and Gummel [9] which is now widely used for the discretization of the semiconductor device equations is utilized. The same approach is used for the discretization of the linear diffusion coefficient. Taking the limits of the linear diffusion coefficient discretization, the discretization for the constant diffusion coefficient is obtained.

Given that the flow of species C_2, is described respectively as

$$J = \lambda exp(\omega C_1)\nabla C_2 \tag{4.11}$$

where C_1 is the other species interacting with C_2, two simplifying assumptions are made. First, C_1 is linearly discretized. Second, the flow of C_2 is constant between grid nodes in a one-dimensional grid. Using techniques used by Scharfetter and Gummel, the constant value of the flux can be extracted. For the exponential model, we can integrate the previous expression between two adjacent grid nodes, *i-1* and *i*, on a one-dimensional domain.

$$\int_{x_{i-1}}^{x_i} \frac{J}{\lambda} exp\left(-\omega\left(\frac{C_{1(i)} - C_{1(i-1)}}{x_i - x_{i-1}}(x - x_{i-1}) + C_{1(i-1)}\right)\right)dx = \int_{x_{i-1}}^{x_i} \frac{d}{dx}C_2 dx \tag{4.12}$$

$$\frac{J}{\lambda}\left(\frac{x_i - x_{i-1}}{(-\omega)(C_{1(i)} - C_{1(i-1)})}(exp(-\beta C_{1(i)}) - exp(-\beta C_{1(i-1)}))\right) = C_{2(i)} - C_{2(i-1)} \tag{4.13}$$

The constant value of the flux can now be extracted.

$$J = \lambda\frac{-\omega(C_{1(i)} - C_{1(i-1)})}{(exp(-\omega C_{1(i)}) - exp(-\omega C_{1(i-1)}))}\frac{C_{2(i)} - C_{2(i-1)}}{x_i - x_{i-1}} \tag{4.14}$$

Equating the flow going from node *i-1* to node *i* to the flow going from *i* to *i+1*, an equation which is linear in the values of C_2 and nonlinear as a function of C_1 is obtained. The previous formulation is customarily extended to multidimensional domain by assuming that the projection of the flux along each line connecting adjacent nodes can be considered constant.

The same steps can de used to derive the constant value of the flux for the constant and linear models.

$$J = \alpha\frac{C_{2(i)} - C_{2(i-1)}}{x_i - x_{i-1}} \tag{4.15}$$

$$J = \frac{\beta(C_{1(i)} - C_{1(i-1)})}{ln\left(\frac{\gamma + \beta C_{1(i)}}{\gamma + \beta C_{1(i-1)}}\right)}\frac{C_{2(i)} - C_{2(i-1)}}{x_i - x_{i-1}} \tag{4.16}$$

4.5 Linear Matrix Assembly

4.5.1 Definition of the Jacobian

The nonlinear equations are solved using the Newton-Raphson method which is given by:

$$C_{n+1}-C_n-\frac{1}{2}h(f(y_{n+1})+f(y_n)) = F(C_{n+1}) = 0 \tag{4.17}$$

$$D(C_{(n+1)(i)})(C_{(n+1)(i+1)}-C_{(n+1)(i)}) = -F(C_{(n+1)(i)}) \tag{4.18}$$

where $D(C_n)$ is the jacobian matrix of the diffusion and bake equations calculated using the values obtained from the previous Newton-Raphson iteration. The model to be used involves two species and is summarized by the equations below.

$$\frac{\partial C_1}{\partial t} = k_1(1-C_1)C_2^m = f_1(C_1, C_2) \tag{4.19}$$

$$\frac{\partial C_2}{\partial t} = \nabla\bullet\ (D_2\nabla\ C_2)+k_2C_2 = f_2(C_1, C_2) \tag{4.20}$$

where C_1 is the concentration of activated sites and C_2 is the concentration of acid. The boundary conditions discussed earlier will be used with the addition of $C_1(t=0)=0$, and $C_2(t=0) = C(x,y,z)$. D_2 is the exponential function discussed earlier with C_j equal to C_1.

The jacobian of the matrix for two species is defined by:

$$D(C_1, C_2) = \begin{bmatrix} \frac{\partial F_1}{\partial C_1} & \frac{\partial F_1}{\partial C_2} \\ \frac{\partial F_2}{\partial C_1} & \frac{\partial F_2}{\partial C_2} \end{bmatrix} \tag{4.21}$$

If the diffusion coefficient D_1 is equal to zero, the matrix $\partial F_1/\partial C_1$ is diagonal. At the same time, matrix $\partial F_1/\partial C_2$ is also diagonal since the generation and recombination are functions of the local values of the variables. Note that matrix $\partial F_2/\partial C_2$ is symmetric as is easily recognized from the discretization, while $\partial F_2/\partial C_1$ is asymmetric. The matrix elements for the first species are derived from the trapezoidal rule equation as follows:

$$F_1(C_{1(n+1)(i)}, C_{2(n+1)(i)}) = C_{1(n+1)(i)}-C_{1(n)}-\frac{1}{2}h(f_1(C_{1(n+1)(i)}, C_{2(n+1)(i)})+f_1(C_{1(n)}, C_{2(n)})) \tag{4.22}$$

$$\frac{\partial F_1}{\partial C_{1(n+1)}} = 1-\frac{1}{2}h((-1)k_1C_{2(n+1)}^{1.42}) \tag{4.23}$$

$$\frac{\partial F_1}{\partial C_{2(n+1)}} = -\frac{1}{2}h(k_1(1 - C_{1(n+1)})(1.42)C_{2(n+1)}^{0.42}) \tag{4.24}$$

The matrix elements for the second species are derived as follows:

$$F_2(C_{1(n+1)(i)}, C_{2(n+1)(i)}) = C_{2(n+1)(i)} - C_{2(n)} - \frac{1}{2}h(f_2(C_{1(n+1)(i)}, C_{2(n+1)(i)}) + f_2(C_{1(n)}, C_{2(n)})) \tag{4.25}$$

$$f_2(C_1, C_2) = \frac{J_{i+\frac{1}{2}} - J_{i-\frac{1}{2}}}{h_x} + \frac{J_{j+\frac{1}{2}} - J_{j-\frac{1}{2}}}{h_y} + \frac{J_{k+\frac{1}{2}} - J_{k-\frac{1}{2}}}{h_z} + k_2C_2 \tag{4.26}$$

where h_x, h_y, and h_z are the dimensions of the control volume. Each control volume is divided into eight subvolumes to handle boundary conditions. Hence, the equations are transformed as follows:

$$g_2(C_1, C_2) = h_yh_z\left(J_{i+\frac{1}{2}} - J_{i-\frac{1}{2}}\right) + h_zh_y\left(J_{j+\frac{1}{2}} - J_{j-\frac{1}{2}}\right) + h_yh_z\left(J_{k+\frac{1}{2}} - J_{k-\frac{1}{2}}\right) + h_xh_yh_zk_2C_2 \tag{4.27}$$

$$F_2(C_{1(n+1)(i)}, C_{2(n+1)(i)}) = h_xh_yh_zC_{2(n+1)(i)} - h_xh_yh_zC_{2(n)} - \frac{1}{2}h(g_2(C_{1(n+1)(i)}, C_{2(n+1)(i)}) + g_2(C_{1(n)}, C_{2(n)})) \tag{4.28}$$

Using the result of the derivation for the spatial discretization and manipulating the exponential terms, we get Equation 4.30 as the flux for the exponential model.

$$J_{i-\frac{1}{2}} = \frac{-\beta(C_{1(i)} - C_{1(i-1)})}{(exp(-\beta C_{1(i)}) - exp(-\beta C_{1(i-1)}))} \frac{C_{2(i)} - C_{2(i-1)}}{x_i - x_{i-1}}\alpha \tag{4.29}$$

$$J_{i-\frac{1}{2}} = exp(\beta C_{1(i-1)})B(-\beta(C_{1(i)} - C_{1(i-1)}))\left(\frac{C_{2(i)} - C_{2(i-1)}}{x_i - x_{i-1}}\alpha\right) \tag{4.30}$$

where $B(x) = \frac{x}{e^x - 1}$ which is the Bernoulli function. The matrix terms can now be derived in terms of the Bernoulli function.

$$\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{2(ijk)}} = exp(\beta C_{1(i-1)})B(-\beta(C_{1(i)} - C_{1(i-1)}))\left(\frac{1}{x_i - x_{i-1}}\alpha\right) \tag{4.31}$$

$$\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{1(ijk)}} = exp(\beta C_{1(i-1)})B'(-\beta(C_{1(i)} - C_{1(i-1)}))(-\beta)\left(\frac{C_{2(i)} - C_{2(i-1)}}{x_i - x_{i-1}}\alpha\right) \tag{4.32}$$

$$\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{1(i-1jk)}} = \left(\frac{C_{2(i)}-C_{2(i-1)}}{x_i-x_{i-1}}\alpha\right)(\beta exp(\beta C_{1(i-1)}))\,(B(-\beta(C_{1(i)}-C_{1(i-1)}))+B'(-\beta(C_{1(i)}-C_{1(i-1)}))) \tag{4.33}$$

The matrix terms for the linear model can be derived the same way.

$$\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{2(ijk)}} = \frac{\beta}{ln\left(\dfrac{\alpha+\beta C_{1(i)}}{\alpha+\beta C_{1(i-1)}}\right)}\frac{C_{1(i)}-C_{1(i-1)}}{x_i-x_{i-1}} \tag{4.34}$$

$$\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{1(ijk)}} = \frac{\beta}{ln\left(\dfrac{\alpha+\beta C_{1(i)}}{\alpha+\beta C_{1(i-1)}}\right)}\frac{C_{2(i)}-C_{2(i-1)}}{x_i-x_{i-1}}\left(1-(C_{1(i)}-C_{1(i-1)})\frac{\beta}{ln\left(\dfrac{\alpha+\beta C_{1(i)}}{\alpha+\beta C_{1(i-1)}}\right)(\alpha+\beta C_{1(i)})}\right) \tag{4.35}$$

$$\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{1(i-1jk)}} = \frac{\beta}{ln\left(\dfrac{\alpha+\beta C_{1(i)}}{\alpha+\beta C_{1(i-1)}}\right)}\frac{C_{2(i)}-C_{2(i-1)}}{x_i-x_{i-1}}\left(-1+(C_{1(i)}-C_{1(i-1)})\frac{\beta}{ln\left(\dfrac{\alpha+\beta C_{1(i)}}{\alpha+\beta C_{1(i-1)}}\right)(\alpha+\beta C_{1(i-1)})}\right) \tag{4.36}$$

A special case is when $\beta C_{1(i)}$ approaches $\beta C_{1(i-1)}$. The limits for Equations 4.34 - 4.36 are shown by Equations 4.37 - 4.38.

$$\lim_{\beta C_{1(i)}\to\beta C_{1(i-1)}}\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{2(ijk)}} = \frac{\alpha+\beta C_{1(i-1)}}{x_i-x_{i-1}} \tag{4.37}$$

$$\lim_{\beta C_{1(i)}\to\beta C_{1(i-1)}}\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{1(ijk)}} = \frac{C_{2(i)}-C_{2(i-1)}}{x_i-x_{i-1}}\left(\frac{\beta}{2}\right) \tag{4.38}$$

$$\lim_{\beta C_{1(i)}\to\beta C_{1(i-1)}}\frac{\partial J_{i-\frac{1}{2}}}{\partial C_{1(i-1jk)}} = \frac{C_{2(i)}-C_{2(i-1)}}{x_i-x_{i-1}}\left(\frac{\beta}{2}\right) \tag{4.39}$$

4.5.2 Matrix Size Reduction

A reduction of the size of the matrix can be obtained by inverting the diagonal matrix $\partial F_1/\partial C_1$ and obtaining ∂C_1 as a function of ∂C_2.

$$\partial C_1 = \left(\frac{\partial F_1}{\partial C_1}\right)^{-1}\left(RHS_1-\frac{\partial F_1}{\partial C_2}\partial C_2\right) \tag{4.40}$$

Replacing now the previous expression in the following equation:

$$\frac{\partial F_2}{\partial C_1}(\partial C_1) + \frac{\partial F_2}{\partial C_2}(\partial C_2) = RHS_2 \qquad (4.41)$$

We obtain:

$$\left[-\frac{\partial}{\partial C_1}F_2\left(\frac{\partial F_1}{\partial C_1}\right)^{-1}\left(\frac{\partial F_1}{\partial C_2}\right) + \frac{\partial F_2}{\partial C_2}\right]\partial C_2 = RHS_2 - \frac{\partial F_2}{\partial C_1}\left(\frac{\partial F_1}{\partial C_1}\right)^{-1} RHS_1 \qquad (4.42)$$

4.6 Verifying the Diffusion Models

Different resist systems exhibit different diffusion behaviors. For example, SAL601 [13] exhibits a Fickian diffusion behavior. APEX-E [21] is reported to show type II diffusion behavior. We verify and extend these studies by focusing on the PEB characteristics of a positive-tone DUV resist used by IBM which is based on the acid catalyzed deprotection of APEX-E and a generic t-BOC resist [32].

Eib et al. [21] reports that the Dill *A* parameter for APEX-E is approximately zero. Hence, an expression for normalized concentration *P* ($P = 1 - M$) can be obtained.

$$P(z,t) = 1 - exp(-CI(z)t) \qquad (4.43)$$

The Dill parameters are measured to be $A = -0.01\mu m^{-1}$, $B = 0.16\mu m^{-1}$, and $C = 0.004 cm^2 mJ^{-1}$ using PEB temperatures ranging from 70C to 100C.

4.6.1 Determination of Equation Parameters

The curves generated by different doses represent different concentrations of acid at the start of the bake. The spacing between the curves can be increased by raising the acid concentration *m* to a power greater than one. If the resist dissolution rate is assumed to be determined by the extent of deprotection, then a higher value of the acid exponent, *m*, will lead to increased resist contrast.

The activated sites, C_2, can be observed with a Fourier transform infrared (FTIR) spectroscopy. An increase in activated sites would decrease the infrared absorbance of the resist. Figure 4.2 illustrates the FTIR characteristics of a fully and patterned exposed Apex-E resist with no reflection. Without diffusion, equations 4.3 - 4.4 can be solved analytically.

$$C_1 = C_{cs}\left(1 - exp\left(-C_{20}^m\left(\frac{k_1}{mk_2}\right)(1 - e^{-mk_2 t})\right)\right) \qquad (4.44)$$

where C_{CS} is a constant representing the total concentration of cross-linking sites and C_{20} is the initial concentration of the photo-generated acid. By using the fully exposed FTIR data and the Levenberg-Marquardt method [29] applied on equation 4.44, the reaction kinetics parameters, k_1, k_2, and m, are determined to be 0.4, 0.0, and 1.6, respectively. By using the patterned exposed FTIR data, α, γ, β, λ, and ω are determined to be 4.0e-16, 1.0e-16, 4.0e-16, 1.0e-16, and 2.5, respectively.

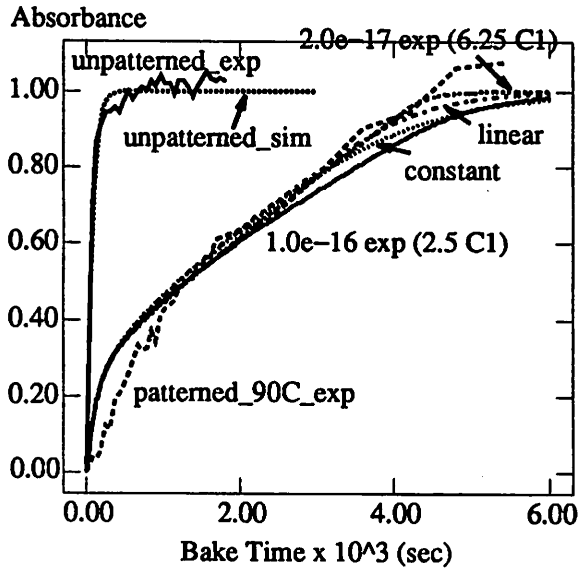

Figure 4.2: FTIR Experimental and Simulation Results

4.6.2 1-D Simulations

The diffusion models are examined using a initial source of acid and making it diffuse in one dimension. Diffusion parameters obtained from the previous section are used for the 1-D simulations. Figures 4.3 and 4.4 illustrate the simulation results for the three diffusion models. Constant and linear diffusion models give similar results while the exponential model is able to sustain the acid front as it propagates through the resist. The steepness of the acid front is determined by the magnitude of the exponential coefficient. Figure 4.4 illustrates a 0.3 um linewidth difference by using a 98% deprotection level for the constant and the exponential model with a high exponential coefficient. A larger linewidth change for the exponential model is also illustrated for results presented in Section 4.6.4. Including a acid loss term (nonzero k_2) affects the magnitude, propagation speed, and curvature of the profiles.

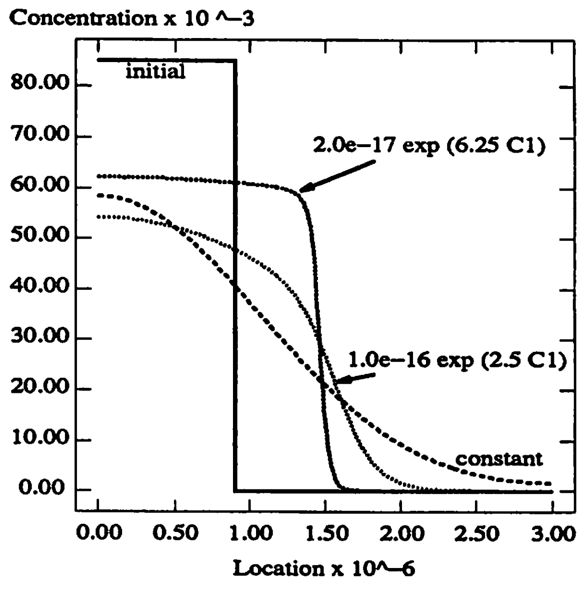

Figure 4.3: Acid Profiles at t = 1000 secs

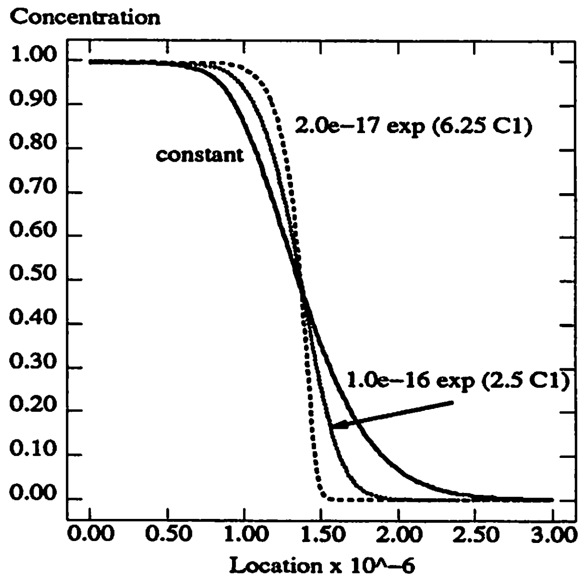

Figure 4.4: Activated Site Profiles at t = 1000 secs

4.6.3 Experimental Results for a RIM PSM with Annular Illumination

To extend optical lithography to smaller features, resolution enhancement techniques are being investigated. One example described by Newmark et al. [23] is the use of rim phase-shifting masks in conjunction with annular illumination. SEM Results of this technique are shown in Figure 4.5 which illustrates experimental results for a 0.44*um* and a 1.08*um* spacing. Although a large depth of focus for varying pitch lines is obtained, the dimensional control of the semi-isolated lines is a problem. Figure 4.5 illustrates that the line-width change between dense and isolated features is 30% or 60nm. Although inaccuracies with the image, and substrate reflections partially explain the proximity effects, a major portion of the difference is still unaccounted for. A possible explanation is diffusion in APEX-E. Newmark et al. [23] show the peak aerial image intensity of the dense lines to be only 0.9 while the peak of the isolated line is 1.9. Since acid concentration is proportional to the peak aerial image intensity as illustrated in section 4.2.1, there is a big difference in acid concentration between the dense and isolated lines.

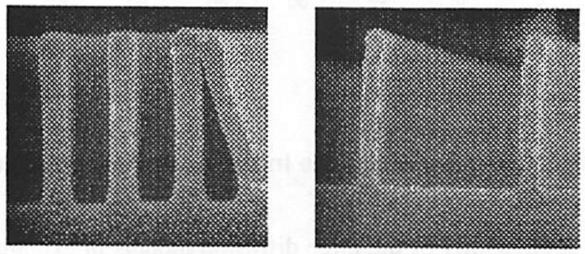
Figure 4.5: 0.44um and 1.08um SEM Cross-Sections

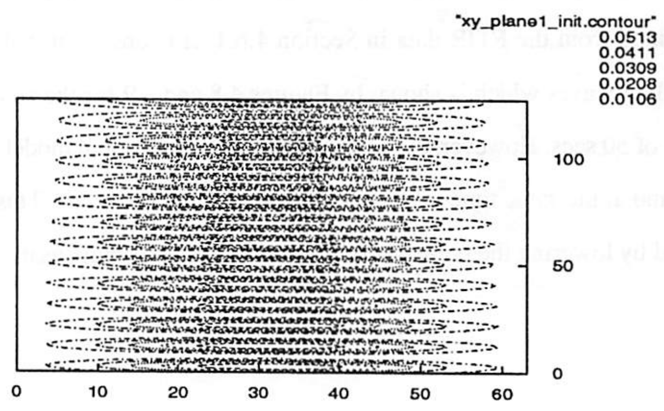

Figure 4.6: Dense Line Initial Acid Concentration

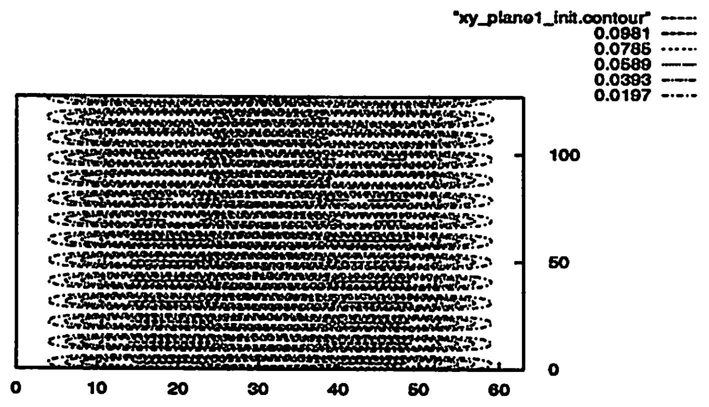

Figure 4.7: Isolated Line Initial Acid Concentration

We investigate the applicability of the three diffusion models in explaining the proximity effects. Intensity distributions are generated by Splat using the mask features described by Newmark et al. [23]. Acid concentration is calculated using Dill's equation and the experimental dose of 16.8 mJ/cm2. Figures 4.6 and 4.7 show the initial acid concentration after exposure for dense and isolated lines respectively. Standing waves result from coherent interference of monochromatic radiation. These data are simulated with parameters obtained from the FTIR data in Section 4.6.1. It is observed that all three models are able to remove the standing waves which is shown by Figures 4.8 and 4.9 for the constant and exponential model after a bake time of 50 secs. However, the parameters of the exponential model may be adjusted to retain the standing waves and at the same time is still consistent with the FTIR data. This is illustrated by Figure 4.10 which is obtained by lowering the pre-exponential coefficient and compensating it with a larger exponential coefficient.

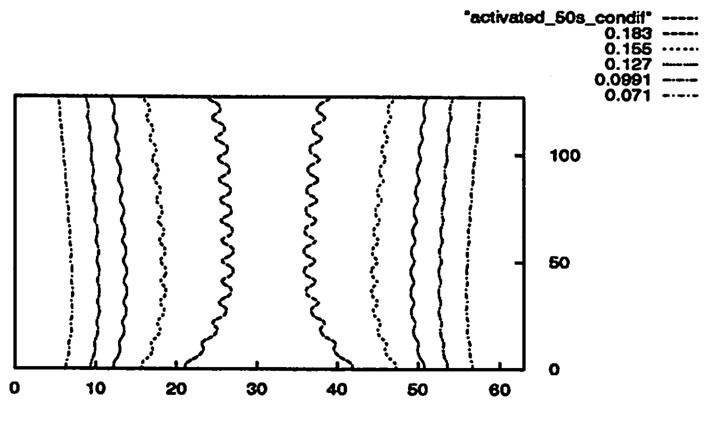

Figure 4.8: Constant Model Activated Sites Contour

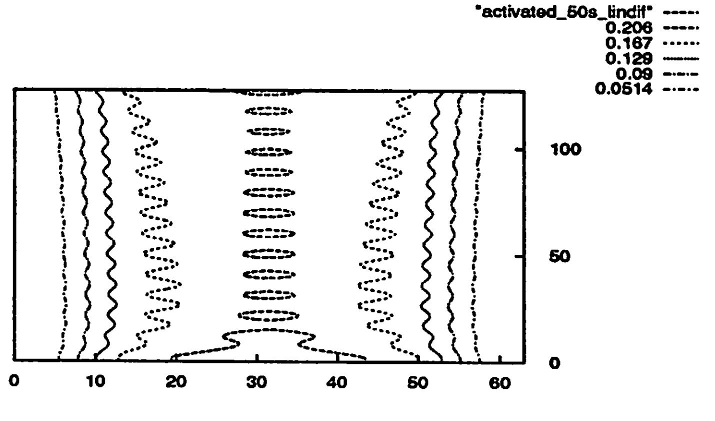

Figure 4.9: Exponential Model with High Pre-exponential Coefficient Activated Sites Contour

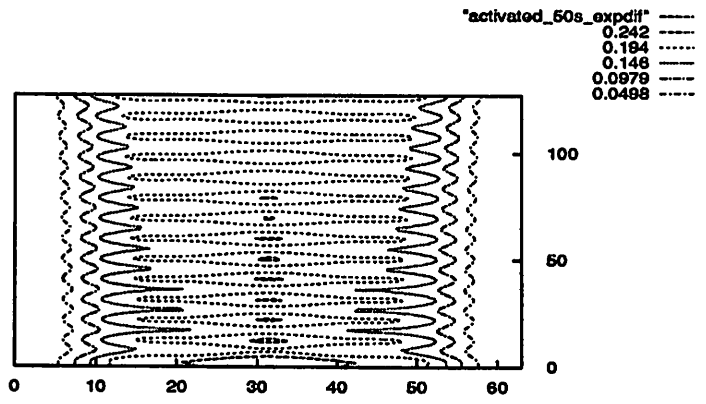

Figure 4.10: Exponential Model with Low Pre-exponential Coefficient Activated Sites Contour

The simulation results give the activated sites for dense and isolated lines. The deprotection level, which is defined as the amount of activated sites needed for the resist to be removed, is determined by setting it to the value of C_2 at the simulation boundary of the linewidth consistent with the 0.44*um* pitch experimental results. This deprotection level is then used to determine the CD of other simulations with a different pitch. Table 4.1 shows that the exponentially activated acid diffusion in Apex-E may explain the observed proximity effects. The lower pre-exponential coefficient is a better model since the standing waves are more diffused compared to the large pre-exponential coefficient model. Initial acid conditions with no standing waves were also examined to decrease the number of grid points necessary for the simulation. However, results obtained do not show proximity effects consistent with the experimental results. The low contour level obtained in determining the CD is probably due to the delay in heating the resist. New mathematical methods in conjunction with experimental results are being developed to further clarify the reaction parameters.

Table 4.1: Critical Dimension Simulation Results at t=50 sec

Diffusion Model	Contour Level	CD (pitch = 0.44 um)	CD (pitch = 1.08 um)
0	0.044	0.210	0.184

Table 4.1: Critical Dimension Simulation Results at t=50 sec

Diffusion Model	Contour Level	CD (pitch = 0.44 um)	CD (pitch = 1.08 um)
$4.0\text{x}10^{-16}$	0.035	0.210	0.0
$2.0\text{x}10^{-17}e^{6.25C1}$	0.030	0.210	0.170
$1.0\text{x}10^{-16}e^{2.5C1}$	0.034	0.210	0.141

4.6.4 Linewidth Measurements

Direct linewidth measurements with the correct deprotection level should further differentiate the diffusion models. Figure 4.11 illustrates experimental linewidth versus post-exposure bake time behavior of different resist systems. Figures 4.12 and 4.13 illustrate simulation linewidth measurements for a generic positive tone t-BOC [32]. There is an observable difference between the constant and the exponential model. With proper reaction kinetics parameters, linewidth measurements through simulations may explain in better detail the diffusion mechanisms of the resist systems in Figure 4.11.

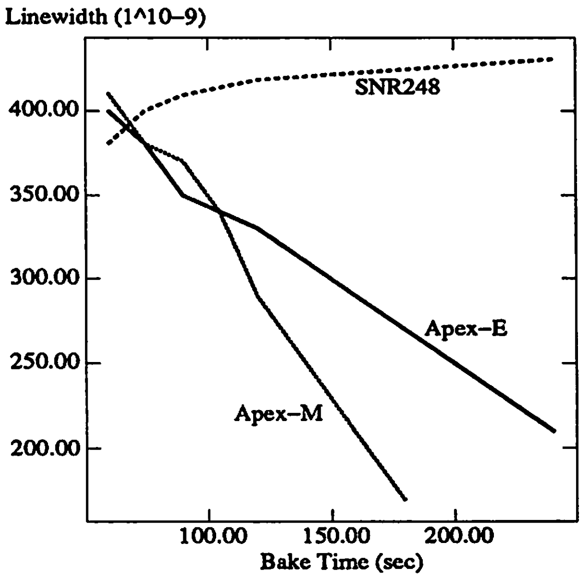

Figure 4.11: Linewidth Measurements for Different Resist Systems

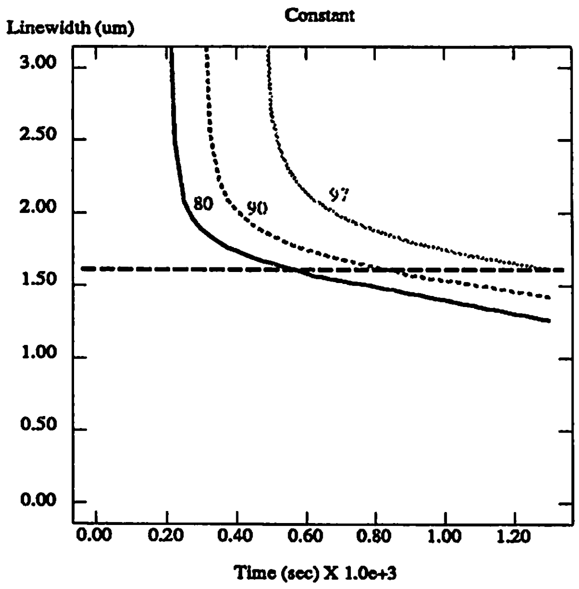

Figure 4.12: Constant Model Linewidth Measurements

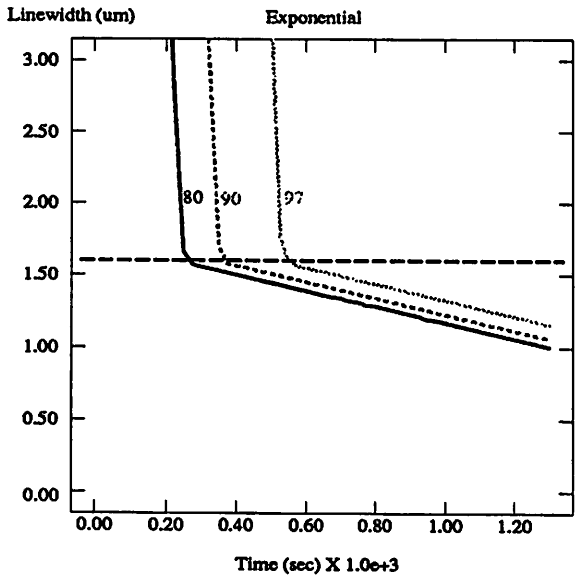

Figure 4.13: Exponential Model Linewidth Measurement

4.7 Applications and Performance Issues

4.7.1 Comparing Implicit and Explicit Methods

We implement a fourth-order Runge-Kutta method to examine the feasibility of an explicit solver. The Runge-Kutta method has been successfully used to solve numerous partial differential equations and has been used by Ferguson [27] to solve a constant acid diffusion model. Using the diffusion parameters obtained in Section 4.6.1, the performance of the Runge-Kutta method is evaluated with the three diffusion models. Since current and future resist materials have different diffusion parameters, the explicit and implicit methods are further examined by increasing and decreasing experimentally determined diffusion parameters.

Tables 4.2 and 4.3 show the time step and CPU time needed for both implicit and explicit methods to achieve the same acid profiles after 500 secs for an initial condition coming from the 25 sec results described in Section 4.6.2. This allows the removal of the steep gradient of the initial condition which would require small initial time steps. Solutions are obtained by using the largest time step possible and still keeping the desired accuracy. The implicit method with no preconditioning is observed to be superior for all the diffusion models in terms of CPU time minimization. Figure 4.14 illustrates that the effectiveness of implicit methods over explicit methods in terms of CPU time minimization increases for increasing diffusivity.

Table 4.2: Variations of Constant Diffusion Model (t = 500 sec)

D Value	Explicit Time Step (sec)	Explicit CPU Time (sec)	Implicit Time Step (sec)	Implicit Iter (No Pre)	Implicit CPU Time (No Pre)	Implicit Iter (Pre)	Implicit CPU Time (Pre)
1.0e-17	25	19.7	200	99	4.7	31	4.6
4.0e-16	0.5	617.5	125	523	14.3	290	18.3
1.0e-15	0.2	2475.0	50	1000	33	675	47.0

Table 4.3: Variations of Exponential Diffusion Model (t = 500 sec)

Exp Coefficient Value	Explicit Time Step (sec)	Explicit CPU Time (sec)	Implicit Time Step (sec)	Implicit Iter (No Pre)	Implicit CPU Time (No Pre)	Implicit Iter (Pre)	Implicit CPU Time (Pre)
0.5	5.0	57.0	200	260	7.7	271	15.7
2.5	1.0	260.0	100	902	24.0	569	33.1
7.0	0.5	625.0	50	1334	39.0	1702	106.7

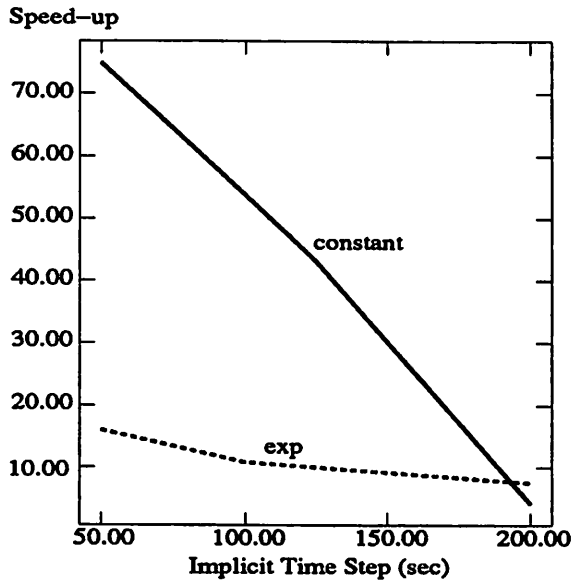

Figure 4.14: Speed-up for Using Implicit Methods

4.7.2 Two and Three Dimensional Simulation

Table 4.4: Machine Scalability (t = 500 sec)

Machine Size	CPU Time (sec)
1K	1031.3
8K	152.3
16K	88.3

Table 4.1 shows the machine scalability of the simulator for the parameters used to generate Figure 4.9. Scalability does not equal 100% since less communication requirements are needed for the smaller machines. Figure 4.15 illustrates the acid contour with a matchhead characteristic. The initial condition shown in Figure 4.7 is used. However, acid concentration at the top 5% simulation region is equated to zero to simulate possible evaporation effects. A more accurate model can be implemented by modifying Equation 4.5 to a nonzero value at the top of the simulation region. These simulations may be helpful in explaining matchhead effects observed by Reuhman-Huisken et al. [25].

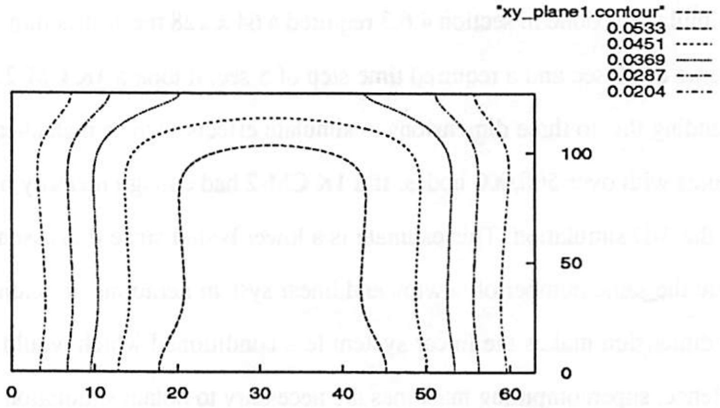

Figure 4.15: Acid Contour with Matchhead Characteristics (t=50 sec)

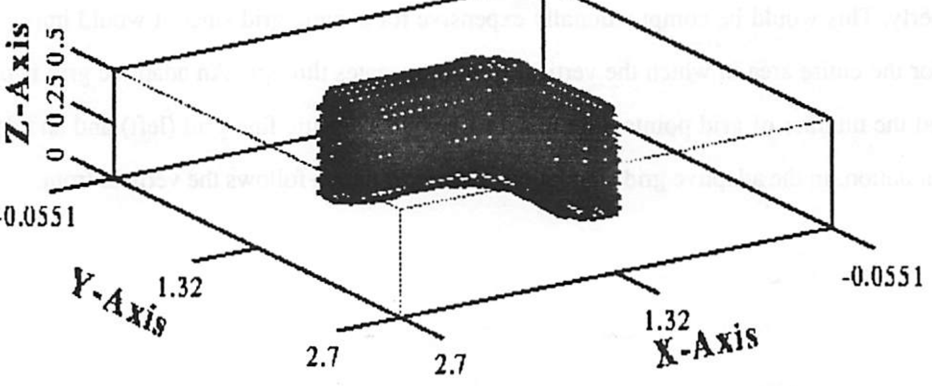

Figure 4.16: 3-D L-Shape Initial Acid Concentration

Three-dimensional simulation of movement and reaction of species in the post-exposure bake of chemically amplified resist systems is invaluable in numerous applications. 3-D end of line simulations may model matchhead effects more accurately since both evaporation and diffusion due to end of line effects can be simulated. Another possible application are holes in resists created by reflective notching [26]. Resist diffusion may be used to remove the holes. Figure 4.16 shows an L-shape initial acid concentration that may also require 3-D simulation due to standing wave effects.

2-D simulations done in section 4.6.3 required a 64 x 128 mesh structure. For a linear diffusion model simulation time of 50 sec and a required time step of 5 sec, it took a 1K CM-2 227 seconds to generate the results. Extending this to three dimensions to simulate effects such as matchhead formation would result in mesh structures with over 500,000 nodes. If a 1K CM-2 had enough memory, it would require more than 4 hours to do the 3-D simulation. This estimate is a lower bound since it is assuming that the 3-D simulation would require the same number of newton and linear system iterations. In reality, the additional complexity of the third dimension makes the linear system less conditioned which would result in more linear solver iterations. Hence, supercomputing machines are necessary to obtain simulation results in a more reasonable amount of time. A 16K node CM-2 and a 64 node CM-5 offer the computation power needed.

4.8 Vertical Fronts and Adaptive Grids

Figure 4.3 shows acid profiles for the three diffusion models. A vertical front is observed for some exponential diffusion simulations. Sturtevant et al. [20] supports the existence of fronts by illustrating experimental data showing a near linear dependence of line space change versus post exposure bake time. Small grid spacings must be used in the vertical front area in order to evaluate the gradient of the acid concentration properly. This would be computationally expensive for a static grid since it would imply a fine grid spacing for the entire area in which the vertical front propagates through. An adaptive grid is proposed to minimized the number of grid points. Figure 4.14 illustrates a static fine grid (left) and an adaptive grid (right) simulation. In the adaptive grid simulation, a fine grid region follows the vertical front.

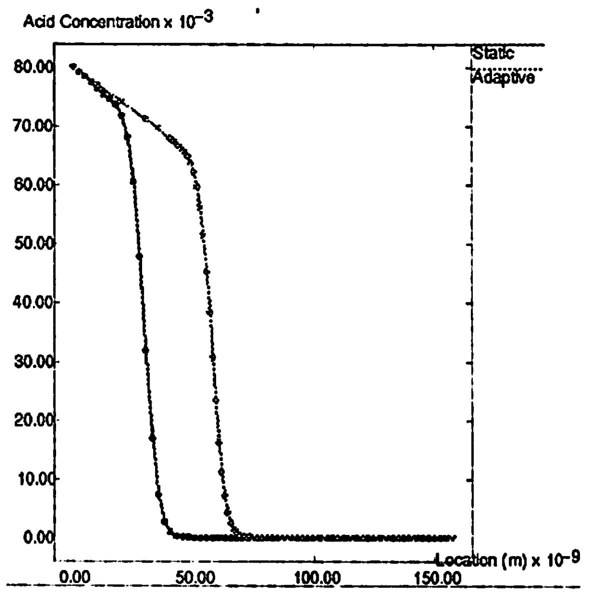

Figure 4.17: Adaptive Grids for Vertical Fronts

The adaptive approach is executed as follows: First, at the beginning of the simulation, the fine grid mesh is located on the starting front. As the simulation progresses, the velocity and size of the front is calculated. The size of the front is determined by marking its boundaries as points that are a certain factor less than the maximum slope or adjacent points that have slopes that only differ by a certain fraction. The velocity of the front is calculated by comparing the location of the maximum slope for two consecutive time points. From these information, grid points are transferred from behind the tail of the front to locations beyond the head of the front. The location of the new points are determined by dividing the distance between currently existing adjacent points by one plus the number of new points specified to be added between existing points.

It is important to place new points at locations with no activity yet in order to accurately linearly interpolate solutions for the new grid points. The interpolation function works as follows: First, determining the unknown nodes with the most number of known neighbors. Second, linearly interpolate these nodes from the values of the known neighbors. Finally, repeat the first step until there are no more unknown nodes.

This scheme will work for fronts that propagate in the direction of one of the coordinate axis. The decrease in the number of grids will depend upon the size and speed of the fronts and the duration of the simulation. Corner effects and standing waves may increase the number of grid points.

The adaptive grid algorithm is tested with a cubic initial acid condition illustrated by Figure 4.18. One corner of the cube is allowed to propagate into the resist in order to observe the 3-D effects on propagating fronts. The width of the front is observed to be 4 times larger compared to a 1-D front using the same parameters (exponential parameter with steep acid vertical front) obtained in Section 4.6.1 for a simulation time of 25 seconds. However, even with the widening of the front, a factor of 2 speed-up is still observed with the use of adaptive grids.

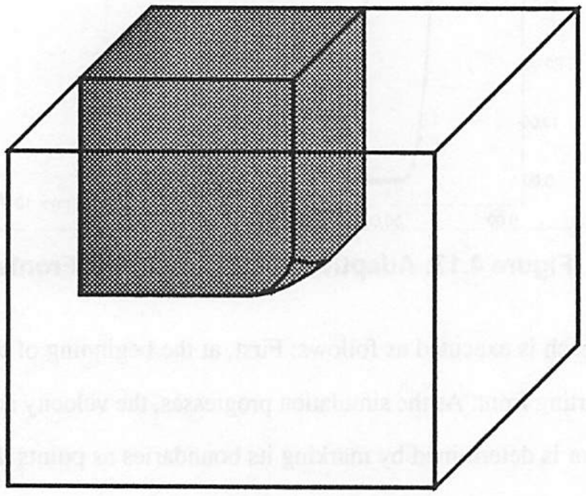
Figure 4.18: 3-D Corner Acid Front

4.9 Extension to Other Models

4.9.1 Generic T-Boc Model

Walraff et al. [30] characterizes two other deep UV resist materials, PTBOCST and PTBMA. Models were proposed for the thermal and acid catalyzed deprotection and extracted rate coefficients using a stochastic kinetics simulator. The time dependent thermal deprotection and the acid-catalyzed deprotection models are summarized by the following equations.

$$\frac{\partial C_3}{\partial t} = k_2 C_1 C_4 - k_{-2} C_3 C_2 \tag{4.45}$$

$$\frac{\partial C_1}{\partial t} = -k_1 C_1 - k_2 C_1 C_4 + k_{-2} C_2 C_3 + \nabla \bullet (D_1 \nabla C_1) \tag{4.46}$$

$$\frac{\partial C_2}{\partial t} = k_2 C_1 C_4 - k_{-2} C_3 C_2 + k_1 C_1 + \nabla \bullet (D_2 \nabla C_2) \tag{4.47}$$

$$\frac{\partial C_4}{\partial t} = -k_2 C_1 C_4 + k_{-2} C_2 C_3 \tag{4.48}$$

where C_1 is the protonated TBOC, C_2 is the protonated HOST, C_3 is the unprotonated TBOC, and C_4 is the unprotonated HOST. k_1 is the thermal deprotection rate coefficient. k_2, and k_{-2} are the protonation reaction rate coefficients. D_1 and D_2 are the diffusion coefficients which may have nonlinear dependencies.

Simulation results are easily generated by using the proposed algorithms presented in Section 4.3. Only the matrix generation algorithms which are basically the derivatives of Equations 4.45 - 4.48 are needed to be added. The same diffusion term matrix generator procedures are used. Figure 4.19 shows that experimental and simulations results agree for flood exposures. FTIR simulation data are generated with constant diffusion model and they fit the experimental data quite well.

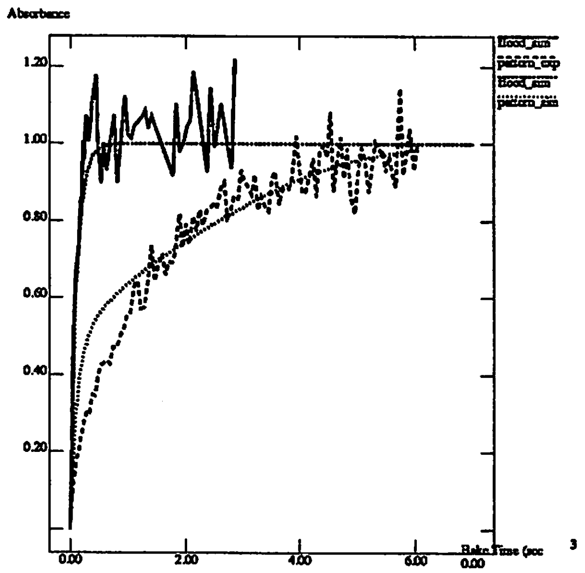

Figure 4.19: Generic T-Boc Experimental and Simulation Results

The addition of two more variables for this new model required twice the memory size. Though each iteration required more CPU time, the proposed solution method works well for both models. The algorithmic behavior is determined by the diffusion terms which couples the matrix rows and makes the linear system of equations harder to solve. Both models appear to create linear systems of equations with similar conditioning characteristics even though the generic t-Boc model has two diffusion terms.

4.9.2 Moving Boundary Due to Volume Shrinkage

The chemically amplified resist simulator has both nonlinear chemical reaction kinetics and simultaneous concentration dependent diffusion. The equations may vary with the type of resist being used. An important phenomena not yet modelled is the volume shrinkage of deprotected areas. Shrinkage of up to

30% are observed in resist systems. The volume shrinkage is needed to be modelled to accurately simulate diffusion. This is a moving boundary problem which will present new challenges for an MPP solver implementation.

The problem may be tackled by initially formulating volume shrinkage equations. This formulation may be influenced by work done on silylation and oxidation. A boundary-fitted curvilinear coordinate system which has been successfully used for device simulation and silicon oxidation [33] may be used to solve this problem. Two variations, an 11 point discretization using Voronoi prism control volume and a 27-point discretization using Voronoi polyhedron control volume can be utilized. Both implementations will handle moving boundaries and, at the same time, retain the same number of grid points and grid connections. Since, grid connections of a boundary-fitted curvilinear coordinate grid structure is equivalent to that of a rectangular grid, there is perfect load and communication balance.

4.10 Summary

The importance of chemically amplified resist systems has necessitated the proper modeling for efficient use of these systems. Two key contributions are presented in this chapter - modeling and a solution method.

Three diffusion models, constant, linear and exponential, have been presented to simulate acid motion within the resist. The diffusion models are differentiated through the examination of 1-D and 2-D simulations. The usefulness of the exponential model has been illustrated by the proximity effect simulations presented in Section 4.6.3 for the Apex-E resist system. The free volume theory [31] is a possible physical explanation since relatively small changes in free volume can lead to a large change in the diffusion coefficient. Standing wave initial conditions are also shown to be important in this section. Wallraff et al. [30] reaction kinetic models have also been successfully implemented and extended with diffusion effects. This model is observed to be accurate for two types of UV resist materials, PTBOCST and PTBMA. The convergence properties for this new model is similar to the Apex-E model. The successful implementation of Ferguson et al. [12] and Walraff et al. [30] models and their extensions show the generality of the simulator. New reaction, diffusion, and boundary models are easily implemented by adding or modifying matrix generation procedures.

The efficient and accurate simulation of the models necessitates the development and implementation of a new discretization scheme and parallel solution algorithms. The nonlinearities of some diffusion models require the careful discretization of the problem through the use of the Scharfetter and Gummel method [15].

Implicit methods are observed to be a lot more efficient than the explicit Runge-Kutta method for solving the equations presented in this chapter. Due to the sparsity of the matrix, the iterative solution methods presented in the previous chapters used for semiconductor device simulation are efficiently used in chemically amplified resist diffusion simulation. The 3-D nature of problems like end of line or L-shape features are shown to require large mesh structures that may even reach half a million nodes. To obtain the solution at a reasonable amount of time, large parallel machines are needed to do the simulations.

References

[1] T. Iwayanagi, T. Ueno, S. Nonogaki, H. Ito, and C. Wilson, "Materials and Processes for Deep-UV Lithography", *Electronic and Photonic Applications of Polymers*, M. Bowden and S. Turner ed., American Chemical Society, Washington, D.C., pp. 109-224, 1988.

[2] C. Wilson, "Organic Resist Materials," *Introduction to Microlithography*, 2nd Edition, L. Thompson, C. Wilson, and M. Bowden ed., American Chemical Society, Washington, D.C., pp. 139-267, 1994.

[3] G. Delzenne, et. al., Advanced Photochem., Vol. 11,1, 1979.

[4] G. Zorpette, "The Power of Parallelism", *IEEE Spectrum*, pp. 28-33, Sept. 1992.

[5] D. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, and A. Sangiovanni-Vincentelli, "A Massively Parallel Algorithm for Three-Dimensional Device Simulation", *IEEE Trans. on CAD*, Vol. 10, pp. 1201-1209, Sept. 1991.

[6] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.

[7] J. Palmer and G. Steele Jr., "Connection Machine Model CM-5 System Overview", *IEEE 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 474-483, 1992.

[8] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*, Springer-Verlag, Wien, Austria, 1984.

[9] D. Scharfetter, and H. Gummel, "Large-Signal Analysis of a Silicon Read Diode Oscillator", *IEEE Trans. Electron Devices*, Vol. 16, pp. 64-77, 1969.

[10] P. Sonneveld, "CGS, A Fast Lanczos-type Solver for Nonsymmetric Linear Systems", *SIAM J. Sci. Stat. Comp.*, Vol. 10, pp. 36-52, Jan. 1989.

[11] Z. Bozkus, S. Ranka, and G. Fox, "Benchmarking the CM-5 multicomputer", *IEEE 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 100-107, 1992.

[12] R. Ferguson, C. Spence, E. Reichmanis, L. Thompson, and A. Neureuther, "Investigation of the Expo-

sure and Bake of a Positive Acting Resist with Chemical Amplification," *SPIE: Advances in Resist Technology and Processing VII*, Vol. 1262, pp. 412-424, 1990.

[13] T. Yoshimura, Y. Nakayama, and S. Okazaki, "Acid-Diffusion Effect of Nanofabrication in Chemical Amplification Resist," *J. Vac. Sci. Technol. B*, Vol. 10, No. 6, pp. 2615-2619, 1992.

[14] W. Hinsberg, S. MacDonald, N. Clecak, C. Snyder, and H. Ito, "Influence of Polymer Properties on Airborne Chemical Contamination of Chemically Amplified Resist," *SPIE Proceedings*, Vol. 1925, pp. 43-52, 1993.

[15] D. Scharfetter, H. Gummel, *IEEE Trans. on Electron Devices*, Vol. ED-16, pp. 66-77, 1969.

[16] T. Toyabe, H. Masuda, Y. Aoki, H. Shukuri, and T. Hagiwara, "Three-dimensional device simulator CADDETH with highly convergent matrix solution algorithms," *IEEE Trans. Electron Devices*, Vol. ED-32, pp. 2038-2044, 1985.

[17] E. Tomacruz, J. Sanghavi, and A. Sangiovanni-Vincentelli, "Algorithms for Drift-Diffusion Device Simulation Using Massively Parallel Processors", *IEICE Trans. on Electronics*, Vol. E77-C, pp. 248-254, Feb. 1994.

[18] Z. Bozkus et.al., *IEEE 4th Symp on the Frontiers of Massively Parallel Computation*, pp. 100-107, 1992.

[19] M. Yeung, D. Lee, R. Lee, and A. Neureuther, "Extension of the Hopkins theory of partially coherence imaging to include thin-film interference effects," *SPIE: Optical/Laser Microlithography VI*, pp. 452-460, Mar. 1993.

[20] J. Sturtevant, S. Holmes, P. Rabidoux, "Post-Exposure Bake Characteristics of a Chemically Amplified Deep-Ultraviolet Resist," *SPIE: Advances in Resist Technology and Processing IX*, Vol. 1672, pp. 114-124, 1992.

[21] N. Eib, E. Barouch, U. Hollerbach, S. Orszag, "Characterization and Simulation of Acid Catalyzed DUV Positive Photoresist," *SPIE*, Vol. 1925, pp. 186-196. 1993.

[22] T. Fedynyshyn, C. Szmanda, R. Blacksmith, and W. Houck, "The relationship between Resist Performance and Diffusion in Chemically Amplified Resist Systems," *SPIE*, Vol. 1925, pp. 2-13, 1993.

[23] D. Newmark, E. Tomacruz, S. Vaidya, and A. Neureuther, "Investigation of Proximity Effects for a Rim Phase-Shifting Mask Printed with Annular Illumination," *SPIE: Optical/Laser Microlithography VII*, 1994.

[24] F. Dill, W. Hornberger, P. Hauge, and J. Shaw, IEEE Trans. on Elec. Dev., Vol. ED-22, No. 7, 1975.

[25] M. Reuhman-Huisken, C. Mutsaers, F. Vollenbroek, and J. Moonen, "Towards an Optimized Image

Reversal Process for Half Micron Lithography," *Microelectronic Engineering*, Vol. 9, pp. 551-556, 1989.

[26] M. Karnett, and M. Sarnoff, "Optimization and Characterization of Single Layer Resist Techniques for 1 um CMOS Production," *SPIE: Optical/Laser Microlithography II*, Vol. 1088, pp. 324-338, 1989.

[27] R. Ferguson, "Modeling and Simulation of Reaction Kinetics in Advanced Resist Processes for Optical Lithography," *Ph.D. Dissertation*, University of California, Berkeley, 1991.

[28] J. Helmsen, M. Yeung, D. Lee, and A. Neureuther, "Sample-3D Benchmarks Including High-NA and Thin-Film Effect," *SPIE*, Vol. 2197, 1994.

[29] D. Marquardt, *J. Soc. Ind. Appl. Math.*, Vol. 11, pp. 431-441, 1963.

[30] G. Walraff, J. Hutchinson, W. Hinsberg, F. Houle, P. Seidel, R. Johnson, and W. Oldham, "Thermal and Acid-Catalyzed Deprotection Kinetics in Deep UV Resist Materials," *International Symposium on Electron, Ion, and Photon Beams*, 1994.

[31] T. Naylor, "Permeation Properties," *Comprehensive Polymer Science: The Synthesis, characterization, Reactions & Applications of Polymers*, Sir G. Allen ed., Pergamon Press, Oxford, 1989.

[32] M. Zuniga, G. Wallraff, E. Tomacruz, B. Smith, C. Larson, W. Hinsberg, and A. Neureuther, "Simulation of Locally-Enhanced 3-D Diffusion in chemically amplified Resists," *J. of Vac. Sci. and Tech.*, pp. 2862-2866, Nov/Dec 1993.

[33] M. Moallemi and H. Zhang, "A General Numerical Procedure for Multilayer Multistep IC Process Simulation", *IEEE Trans. on CAD*, Vol. 13, pp. 1379-1390, Nov. 1994.

CHAPTER 5

Conclusions and Future Work

5.1 A Summary of Contributions

The work presented in the previous chapters have been published in several conferences and journals. The contributions are summarized as follows:

CM-2 rectangular grid drift-diffusion device simulator [1] - Based on a CGS linear solver with a partitioned natural ordering preconditioner, a new massively parallel algorithm for 3-D device simulation is presented. Compared to a sequential machine running the best sequential algorithm, the CM-2 achieves supercomputer performance for problems with more than 15,000 grid nodes.

CM-2 multigrid approach [2], [3]- A multigrid discretization has been developed to provide a framework to perform a block Newton iteration. Three variations of a block Newton iteration are shown to be effective in generating a good initial guess for the device simulator without having any knowledge of the device structure and the operating region. A factor of two speed-up is observed for large MOS and BJT simulations.

CM-5 rectangular grid drift-diffusion simulator [2], [3]- A new preconditioner called the block partitioned natural ordering for a CM-5 drift-diffusion simulator gives a robust and efficient iterative linear solver. It is observed that preconditioners that maintain coupling between nodes give the best results. Also, not having the same cut points for forward and backward substitution is important for producing converging preconditioners. A 128 node CM-5 is observed to provide a vector supercomputer performance.

Capacitance of Silicon Pixel Detectors [4] - A study of the capacitance of pixel detectors to be used as tracking devices for high energy physics experiments. The pixel capacitance matrix plays an important role in system design issues such as preamplifier matching and cross-talk among pixels. A good agreement between simulations and measurements of pixel capacitance is found. The CM-2 drift-diffusion device simulator is currently being used to improve the design of the silicon pixel detectors.

CM-2 chemically amplified resists diffusion simulator [5], [6], [7] - A massively parallel framework for accurate, efficient, and convergent simulation of diffusion models is presented. This involves the use of the solution process used for the CM-2 drift-diffusion simulator. New ideas such as the Scharfetter and Gummel discretization method for acid discretization is utilized along with the use of adaptive grids which were found to improve the simulator performance. Experimental and simulation results on proximity effects show the applicability of an exponential diffusion model. Implicit methods are shown to be more efficient compared to explicit methods for all the diffusion models. Three-dimensional simulation of resist systems are observed to be very computationally intensive but can be simulated in a reasonable amount of time on MPPs.

CM-5 irregular grid drift-diffusion device simulator [8], [9]- A device simulator that uses an irregular grid generated by OMEGA is presented. The problem is solved by initially partitioning the grids using the geometrical, topographical, and spectral partitioning heuristics. Communication scheduling is then done through the repeated use of the maximal nonbipartite matching heuristic. Finally, the preconditioning problem is solved by experimenting with several variations of ILU computations and of the forward and backward substitution algorithms. For large problems, a 60% efficiency for CGS with no preconditioning and 50% efficiency for the solution of the matrix. CGS with processor ILU and automatic magnitude threshold fill-in preconditioning is used for the CM-5 while CGS with ILU is used for PILS.

5.2 Conclusions

In this thesis, the use of parallel processors for device and process simulation has been investigated. In chapter 2, three-dimensional device simulations are observed to be very computationally intensive even with vector supercomputers. The main computational task is the solution of the sparse linear system of equations which may have more than a million equations. The efficiency of the iterative linear solver is determined by the preconditioning scheme. The partitioned natural ordering is observed to give the best results for the CM-2 in terms of CPU time minimization. A preconditioner called the block partitioned natural ordering for a CM-5 drift diffusion simulator gives a robust and efficient iterative linear solver. It is observed that preconditioners that maintain coupling between nodes give the best results. Also, not having the same cut points for forward and backward substitution is important for producing converging preconditioners. A multigrid discretization has been developed to provide a framework to perform a block Newton iteration. Three variations of a block Newton iteration are shown to be effective in generating a good initial guess for the device simulator without having any knowledge of the device structure and the operating region. The parallel algorithms are successfully used to simulate silicon pixel detectors. Three dimensional capacitance simulations which match experimental results are observed to be significantly different from two dimensional simulations. 3-D long range pixel coupling are observed to be amplified due to the blocking strips.

In chapter 3, a parallel irregular grid drift-diffusion device simulator has been presented. A comparison between the best sequential algorithm and the proposed parallel algorithm has revealed a parallel efficiency that exceeds 50% for large problems. Perfect node load balance is observed to be the most important partitioning parameter. In trying to improve the simulator, explicit methods are shown not be as efficient compared to implicit methods for the drift-diffusion equations. An adaptive grid algorithm is described along with the difficulties in its implementation. The proposed irregular grid algorithm is also used in solving the hydrodynamic and circuit noise equations. It is observed that the hydrodynamic and drift-diffusion matrices have comparable conditioning. Also, for large hydrodynamic and circuit noise problems, good parallel efficiency are expected to be observed.

In chapter 4, three diffusion models have been presented to accurately simulate the motion of acid within the resist. The nonlinearity of some diffusion models require the careful discretization of the problem through the use of the Scharfetter and Gummel method. The usefulness of the exponential model has been illustrated by the proximity effect simulations presented in Section 4.6.3 for the Apex-E resist system. Standing wave initial conditions are also shown to be important in this section. Implicit methods are observed to be a lot more efficient than the explicit Runge-Kutta method for solving the equations presented in this chapter. Due to the sparsity of the matrix, the iterative solution methods presented in the previous chapters used for semiconductor device simulation are efficiently used in chemically amplified resist diffusion simulation. The 3-D nature of problems like end of line or L-shape features require the use of large mesh structures that may even reach half a million nodes. To obtain the solution at a reasonable amount of time, large parallel machines are needed to do the simulations. The solution process is also shown to be effective for two other types of UV resist materials, PTBOCST and PTBMA. New models are easily implemented by adding a new matrix generation procedure. The convergence properties for this new model is similar to the Apex-E model.

A framework for solving PDEs can be formulated. First, the type of mesh structure (rectangular, triangular, terminated line, etc.) and grid generation algorithm has to be chosen. Each type has its own advantages and disadvantages with respect to performance issues and difficulty of implementation. After discretization, the grids have to be partitioned to different processors with load and communication balance as key goals. Several algorithms are available such as the geometrical, topographical, and spectral partitioners. Based on the communication pattern and structure of the communication network, each partition is then physically mapped to a specific processor. With the required algorithms for the solution process chosen, a communication scheduling is then done.

Efficient parallel PDE solvers are usually characterized by good load balancing and low communication requirements. Efficiency is defined as minimizing the CPU time required to solve the problem. Hence, processor utilization may be a good indicator of efficiency. However, the best sequential algorithm may not necessarily be the best parallel algorithm. As shown in Chapter 2 for example, red-black ordering for preconditioning may give good processor utilization but may not converge to the right solution. MFLOPs ratings therefore for parallel solution of TCAD equations are not good indicators of efficiency.

Besides the framework, several other presented algorithms are of general use. First, a linear system solver is needed for solving any linearized equations that are usually products of stiff PDEs. Significant research is being done on improving the iterative solver itself. Creativity is also needed in the design and implementation of the preconditioner. The success of the iterative algorithm and the preconditioner is very problem specific. Second, multigrid methods can be used effectively to improve the initial guess of the Newton-Raphson iteration. Third, adaptive grid methods can be used to optimize the use of computational resources.

To summarize, it is observed that MPPs are very useful to TCAD problems. Future peak performance improvement of MPPs over vector supercomputers will make MPPs more attractive for TCAD problems. Since peak performance can only be translated into sustained performance by software, success of MPP TCAD applications will depend significantly on the design and implementation of algorithms. An applications developer would also need to take into account the type of architecture, speed, memory, communication bandwidth, and other hardware considerations in order to produce an efficient parallel TCAD implementation. The presented framework and algorithms should give a TCAD MPP applications developer direction and flexibility in implementing an efficient MPP application.

5.3 Future Work

Several areas of future work have been identified in the previous chapters. In chapter 2, curvilinear coordinate systems may be used as an alternative to rectangular grids to allow the modeling of arbitrarily shaped boundaries. Since the connectivity between grids is identical to a rectangular mesh, algorithms used for parallel rectangular grid may be applicable for this discretization. Parallel rectangular grids simulators also have a lot of potential applications. For example, a more detailed study of silicon detectors would be of significant interest to the nuclear physics community. Other applications such as parasitic MOS and DRAM devices are of great importance due to the constant reduction in feature size.

In chapter 3, a parallel adaptive grid is motivated to be a good extension of the parallel static grid simulator. Adaptive grids are very useful since they modify mesh structures during simulation to compensate for unpredictable and changing fields and current. The application of static grid algorithms to the hydrodynamic models is another possible extension since drift-diffusion simulators lose their validity as feature sizes shrink. A third possible extension is the use of the same algorithms for noise simulation of nonlinear dynamic circuits. Time-domain non-Monte Carlo simulation for nonlinear dynamic circuits with arbitrary excitations require the solutions of large linear systems of equations which may be significantly accelerated with parallel machines. Chapter 3 also motivates the possibility of using workstation cluster as a parallel processing environment. Workstation clusters require more complicated code to hide the latency of sending messages through the network. However, obtaining good efficiency on this platform may have a more significant impact compared to MPP based TCAD tools since workstation clusters are very accessible to most engineers.

In chapter 4, more accurate modeling can be done through the addition of nonzero derivative boundary conditions to model evaporation. New species can be introduced by adding more variables and equatiions. These new species may be modeled to decrease acid concentration as they diffuse through the resist area. The solution algorithms are shown to be easily adaptable to other resist systems. Only the matrix generations routines are needed to be added. The same diffusion term matrix generator procedures are used. A possible extension of the work is the automatic generation of matrix terms given a set of reaction and diffusion equations. This will require routines that evaluate the derivative of the equations the user is supplying. Another possible extension is the use of the curvilinear coordinate system to model moving boundaries due to volume shrinkage. Curvilinear coordinates systems have already been used successfully for the sequential solution of moving boundary problems resulting from silicon oxidation [10].

Reference

[1] D. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, A. Sangiovanni-Vincentelli, "A Massively Parallel Algorithm for Three Dimensional Simulation", *IEEE Trans. on CAD*, Vol. 10, pp. 1201-1209, 1991.

[2] E. Tomacruz, J. Sanghavi, A. Sangiovanni-Vincentelli, "Algorithms for Drift-Diffusion Device Simu-

lation Using Massively Parallel Processors," VPAD Proceedings, pp. 20-21, 1993.

[3] E. Tomacruz, J. Sanghavi, A. Sangiovanni-Vincentelli, "Algorithms for Drift-Diffusion Device Simulation Using Massively Parallel Processors," *IEICE Trans. on Electronics*, Vol. E77-C, pp. 248-254, Feb. 1994.

[4] L. Bosisio, F. Forti, E. Tomacruz, "Measurement and Tridimensional Simulation of Silicon Pixel Detector Capacitance", *IEEE Nuclear Science Symposium*, Vol. 1, pp. 338-342, 1993.

[5] E. Tomacruz, M. Zuniga, R. Guerrieri, A. Neureuther, A. Sangiovanni-Vincentelli, "3-D Diffusion Models for Chemically-Amplified Resists Using Massively Parallel Processors", *SISDEP Proceedings*, pp. 109-112, 1993.

[6] M. Zuniga, G. Wallraff, E. Tomacruz, B. Smith, C. Larson, W. Hinsberg, and A. Neureuther, "Simulation of Locally-Enhanced 3-D Diffusion in chemically amplified Resists," *J. of Vac. Sci. and Tech.*, pp. 2862-2866, Nov/Dec 1993.

[7] D. Newmark, E. Tomacruz, S. Vaidya, and A. Neureuther, "Investigation of Proximity Effects for a Rim Phase-Shifting Mask Printed with Annular Illumination," *SPIE: Optical/Laser Microlithography VII*, 1994.

[8] J. Sanghavi, E. Tomacruz, and A. Sangiovanni-Vincentelli, "Massively Parallel Device Simulation Using Irregular Grids", *Nupad Proceedings*, pp. 141-144, 1994.

[9] E. Tomacruz, J. Sanghavi, and A. Sangiovanni-Vincentelli, "A Parallel Iterative Linear Solver for Solving Irregular Grid Semiconductor Device Matrices," *Supercomputing '94*, pp. 24-33, 1994.

[10] M. Moallemi and H. Zhang, "A General Numerical Procedure for Multilayer Multistep IC Process Simulation", *IEEE Trans. on CAD*, Vol. 13, pp. 1379-1390, Nov. 1994.