CONCURRENT SOLUTIONS OF LARGE SPARSE LINEAR SYSTEMS

TONGSHENG ZHENG

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

Concurrent Solutions of Large Sparse Linear Systems

by

Tongsheng Zheng

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

September 1998

St. John's Canada

Abstract

Computer-aided circuit analysis, or circuit simulation, is widely used in the area of circuit design. Circuit simulation programs, e.g. SPICE, create and solve systems of differential equations which describe the analyzed electronic circuit. The systems of differential equations are converted into nonlinear algebraic equations and solved through a sequence of linear approximations to the nonlinear equations.

Parallel processing is a promising way to improve the performance of circuit simulation programs. Several attempts to port sequential codes into equivalent ones for shared-memory architectures have been reported in the literature. However, with the increasing popularity of message-passing systems, our project aims at the parallelization of a circuit simulation program on a network of workstations. A domain decomposition approach was implemented through a master-slave model on a cluster of SUN stations. The experiments show speedups over up to 8 workstations. This presentation discusses the implemented algorithm and provides an overview of some performance results.

Acknowledgments

This thesis would not have been finished without the guidance of my supervisor Dr. W.M. Zuberek. I am forever grateful to him for providing the knowledge and encouragement to help me pursue my graduate studies.

I also wish to thank Mr. Nolan White and Mr. Michael Rendell for helping me with the implementation part of this thesis. Their suggestions for my work were very valuable. Ms. Floarea Suciu deserves my acknowledgment for providing related information to my work. I would like to thank Ms. Elaine Boone for her continuous assistance throughout my studying in the Department of Computer Science, Memorial University of Newfoundland.

Special thanks go to my family and friends. Their emotional support and understanding have made my stay more bearable and will be the striking force in my life.

Contents

1 Introduction **1**

2 Dynamic Behavior of Programs **4**

2.1 Run-time program profile 5

2.2 SPICE-PAC profile 8

3 Parallel and Distributed Methods **10**

3.1 Direct and iterative solutions of linear systems 11

3.2 Parallel direct linear solvers 15

3.2.1 Shared-memory architectures 15

3.2.2 Distributed-memory architectures 18

3.2.3 Clusters of workstations 21

3.3 Parallel iterative solvers 23

3.4 Parallel hybrid solvers 24

4 The Domain Decomposition Method **26**

4.1 An overview of the domain decomposition approach 28

4.2 The Master-Slave model 30

4.3 Matrix partitioning 31

4.3.1 Graph representation of sparse matrices 33

4.3.2 The minimum degree algorithm 35
4.3.3 Improvements to the minimum degree algorithm 41
4.3.4 Finding the separator 46
4.4 Summary 54

5 Implementation. Results and Discussion **55**
5.1 Data structures representing sparse matrices 55
5.2 Performance 56

6 Conclusions **62**

Bibliography **63**

Appendices **69**

A The Results of the SPICE-PAC Profile **69**
A.1 The results from "test-adder" 69
A.2 The results from "test-adder-2" 71
A.3 The results from "test-add-2" 73
A.4 The results from "test-inv-1" 75
A.5 The results from "test-osc-1" 79

B Circuit Descriptions **83**
B.1 Circuit "test-adder" 83
B.2 Circuit "test-adder-2" 84
B.3 Circuit "test-inv-1" 84
B.4 Circuit "test-osc-1" 85

List of Tables

2.1 Profile of a program. 6

2.2 The input circuits. 8

5.1 Sparse matrix format. 56

5.2 Description of tested linear systems. 56

5.3 Partitioning of MATRIX1 and MATRIX2. 59

List of Figures

3.1 Shared-memory architecture. 16

3.2 Distributed-memory architecture (1) and hybrid architecture (2). 19

4.1 Speedup of a p processor system with different workload distributions W. 28

4.2 An unsymmetric matrix and its directed graph. 33

4.3 A symmetric matrix and its directed graph. 33

4.4 A matrix and its rooted tree. 34

4.5 Elimination sequence for the matrix from Figure 4.3. 35

4.6 A structurally symmetric matrix pattern and its graph representation. . . 37

4.7 Elimination graphs after removing node 10 and node 6. 38

4.8 Pattern of the filled matrix F of $P_1AP_1^T$. 39

4.9 Elimination graphs after removing node 1 and node 2. 40

4.10 Pattern of the filled matrix F of $P_2AP_2^T$. 41

4.11 Elimination graph after first loop of multiple mass elimination. 45

4.12 Pattern of the filled matrix F of $P_3AP_3^T$. 45

4.13 Result of removing node separator $S = \{1, 7, 9\}$. 47

4.14 Result of removing node separator $S = \{1, 4\}$ in graph Figure 4.6. 47

4.15 Graphs $G(A)$ and $G(F)$. 49

4.16 Generating the elimination tree using depth-first search. 50

4.17 Graph representation of the reordered matrix in Figure 4.12 and the generated elimination tree. 53

4.18 Partition of the matrix. 53

5.1 Performance of test problems on network of workstations. 58

5.2 Computation and communication time. 61

Chapter 1

Introduction

Until the appearance of integrated circuits, computational methods found little use in the analysis and design of electronic circuits. A slick designer could synthesize the relatively simple circuits with only minimal computational effort, set them up on a bench, take measurements, make modifications, and quickly arrive at the final versions.

The situation changed dramatically with the arrival of integrated circuits, containing large functional blocks with thousands of transistors on one chip. Obviously, such designs cannot be carried out by experimenting on a bench. Computer technology and numerical mathematics have had profound impact on computer-aided circuit analysis and design.

Computer-aided circuit analysis, or circuit simulation, is the process of building and solving a system of ordinary differential equations that model an electronic circuit. The system of equations is built using Kirchhoff's laws to provide mathematical models of individual devices in the circuit. The systems of differential equations are usually solved by "direct methods" using backward differentiation formula to convert the system of differential equations into a system of nonlinear algebraic equations. This system of nonlinear equations is typically solved using Newton's method, i.e. using linear approx-

imations of nonlinear equations in consecutive points of an iterative process converging to the solution. The systems of linearized equations are usually solved by LU factorization followed by forward and backward substitutions in the factorized matrices.

Since the matrices of coefficients for the linearized systems of equations are very sparse (each element is normally connected with only 3 to 5 other elements), the computation time and memory requirements can be reduced significantly if LU factorization can preserve this sparsity. This requirement, however, may conflict with the optimal pivoting strategy. So usually a compromise must be reached between the accuracy requirements (implied by the pivoting strategy) and the preservation of sparsity. Since the distribution of non-zero elements in the coefficient matrix is implied by the circuit topology (which is constant), the order of elimination of consecutive variables needs to be determined only once.

Conventional circuit simulations, such as SPICE, have been widely used because of their general applicability and high accuracy. Many efforts have been made to successfully improve their performance on general-purpose computers. However, rapid growth in circuit integration requires the performance of circuit simulators to improve as rapidly.

One of the popular approaches to radically improve the performance of circuit simulation is the relaxation method, as adopted in RELAX, SPLICE and NOTIS. However, this approach often results in deterioration of accuracy and convergence problems for circuits with strong feedback. An alternative approach is to use parallel processing. Several attempts used shared memory machines for parallel circuit simulation as the transformation of traditional sequential code into an equivalent one for a shared memory architecture is rather straightforward. Recently, however, message-passing systems

have been gaining popularity because of easily available high-performance workstations. The concept of "virtual parallel machines", i.e., clusters of (possibly heterogeneous) high-performance machines connected by a fast network, is an interesting alternative to traditional supercomputing systems. Several message-passing libraries have been developed (PVM and MPI are just two examples) and ported to many computing platforms to facilitate parallel processing on such clusters of computers.

The purpose of this thesis is to find an efficient way to port a large sequential circuit simulation program, SPICE-PAC, onto a network of workstations. SPICE-PAC is upward compatible with the SPICE-2G6 program, i.e., it has some new features added to the original SPICE simulator [34]. In parallelization of sequential programs, quite often it is not obvious which part of the program has the most important effect on the program's execution time. Therefore a profile of the program can be obtained in order to find out its most time-consuming parts. After profiling SPICE-PAC, two subroutines were selected to be the most promising candidates for parallelization, SPPDCD and SPPDCS. SPPDCD decomposes the (sparse) matrices, while SPPDCS solves the decomposed system of linear equations.

Chapter 2 describes this profiling process on SPICE-PAC program.

Chapter 3 reviews the research work on solving large sparse linear systems. The traditional direct and iterative method are introduced. Different approaches for exploiting parallelism of sparse matrix computation are explained and compared.

Chapter 4 is devoted to presenting a domain decomposition method which is more suitable for solving linear equations on a network of workstations. The method is realized through a master-slave model. The implementation result and conclusions are shown in Chapter 5 and Chapter 6, respectively.

Chapter 2

Dynamic Behavior of Programs

Computer aided circuit simulation programs are tools for design and verification of electronic circuits at the transistor level. Since a large circuit may have several thousands of transistors, the simulation requires considerable resources. At this point, parallel and distributed computing methods are naturally considered to save computation time and solve the space problem. Vectorization of the popular SPICE–like simulator (SPICE2-S) was implemented and observed with no vector speedup [9]. The reason is that the algorithms and data structures are not suitable for vectorization, so special versions of SPICE2 for vector processors have been developed [9]. Our interest is to parallelize the original SPICE-PAC program. It is not possible to globally parallelize or vectorize the whole program without modifying the code. Therefore, we concentrate on the most time consuming part of the program and try to parallelize it in a distributed environment.

The main tasks for the SPICE-based programs are to set up a system of equations from the circuit description and to compute solutions of this system in the time domain [9]. Using the specific device equations leads to a coupled system of implicit nonlinear ordinary differential equations and linear equations. The unknowns in the system are basically the node voltages, and the number of equations is approximately equal to

the number of circuit nodes. The nonlinear system $F(X) = 0$ is solved by Newton-Raphson iteration which requires the computation of the device characteristics and their derivatives and the solution of a system of linear equations at each Newton iteration.

Feldmann [9] gave a brief description for the inner transient analysis part of the circuit simulator. The construction and computation of linear systems take 90 percent or even more of the total transient analysis time. In Feldmann's experiment, for small and medium size circuits, matrix construction time is dominating. For very large circuits, the nonlinear time for linear system solving becomes dominant.

In the next section, we will describe our work on profiling the SPICE-PAC code in order to find out the most time consuming part of the program.

2.1 Run-time program profile

To evaluate the dynamic behavior of the program, we use the system utility **gprof** on workstations. **gprof** produces a call graph showing the execution profile of a program. The profile data is taken from the profile file (gmon.out by default) which is created by programs compiled with **-pg** option. The **gprof** profiling tool reads the symbol table in the specified object file, correlating it with the call profile file. To produce call graph data, it needs three steps:

1. Compile the program with **-pg** option;

2. Execute the program to produce a data file (gmon.out by default);

3. Run **gprof** on the data file;

The profile contains a listing with the total execution time and call counts for each of the functions in the program, sorting by decreasing time. A section of a sample output

index	%time	self	descendants	called/total called+self called/total	parents name children	index
		0.00	65.91	1/1	main	[2]
[1]	100.0	0.00	65.91	1	sppac_	[1]
		0.00	65.19	1/1	spicer_	[3]
		0.00	0.40	1/1	spicec_	[23]
		0.00	0.32	1/1	spicea_	[29]
		0.00	0.00	1/1	outres_	[74]
		0.00	0.00	1/1	extime_	[85]
		0.00	0.00	1/1	spicej_	[98]
		0.00	0.00	1/1	otpfil_	[107]
		0.00	0.00	1/1	spicem_	[108]
...	...	...	...	...	...	...
		0.00	65.19	1/1	sppac_	[1]
[3]	98.9	0.00	65.19	1	spicer_	[3]
		0.02	65.06	1/1	spptra_	[4]
...	...	...	...	...	...	...

Table 2.1: Profile of a program.

from SPICE-PAC profile is shown in Table 2.1 (The more complete files can be found in the appendix). Each section of the table contains the information of the function with indicated index and its parent and children.

1. **function entries:**

   - *index*: the index of the function in the call graph listing.
   - *%time*: the percentage of the total time of the program accounted for by this function and its descendants.
   - *self*: the number of seconds spent in this function itself.
   - *descendants*: the number of seconds spent in the descendants of this function on behalf of this function.

- *called*: the number of times this function is called (other than recursive calls).
- *self*: the number of times this function calls itself recursively.
- *name*: the name of the function.
- *index*: the index of the function in the call graph listing.

2. **parent listings:**

   - *self*: the number of seconds of this function's self time which is due to calls from this parent.
   - *descendants*: the number of seconds of this function's descendant time which is due to calls from this parent.
   - *called*: the number of times this function is called by this parent. This is the numerator of the fraction which divides up the function's time to its parents.
   - *total*: the number of times this function was called by all of its parents. This is the denominator of the propagation fraction.
   - *parents*: the name of this parent.
   - *index* : the index of this parent in the call graph listing.

3. **children listings:**

   - *self*: the number of seconds of this child's self time which is due to being called by this function.
   - *descendants*: the number of seconds of this child's descendant's time which is due to being called by this function.

Name	*Size*	*Description*
test-add-2	227	2-bit all NAND gate binary adder
test-adder	115	1-bit all NAND gate binary adder
test-adder-2	115	1-bit all NAND gate binary adder
test-inv-1	9	MOS inverter
test-osc-1	26	oscillator

Table 2.2: The input circuits.

- *called*: the number of times this child is called by this function. This is the numerator of the propagation fraction for this child.
- *total*: the number of times this child is called by all functions.
- *children*: the name of this child.
- *index*: the index of this child in the call graph listing.

The time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called.

2.2 SPICE-PAC profile

In order to profile SPICE-PAC, the whole program was recompiled with option **-pg** on a DEC workstation. A few changes were made to customize the program because of the different version of FORTRAN compiler than before. We have tested several circuit input files which differ in both structure and size. Table 2.2 gives a brief description of these input circuits.

From the profile result, as shown in Appendix A, the execution of subroutine SPPLDM is the most time consuming part. SPPLDM includes SPPBJT, SPPMOS etc., which "load" matrix. However, as the size of the input circuit increases, the subroutines

which solve sparse linear equations (SPPDCD and SPPDCS) are taking more computation effort. So, for those very large circuit simulation problems, SPPDCD and SPPDCS are the most promising candidates for parallelization.

SPPDCD decomposes the coefficient matrices and SPPDCS solves the decomposed system of linear equations. The main objective of this project is to find an efficient concurrent linear solver to take the place of the two SPICE-PAC subroutines, so as to gain a better performance of the simulation program.

Chapter 3

Parallel and Distributed Methods

The dynamic behavior of many continuous–time systems can be described by a system of differential equations. For the purpose of numerical simulation, these systems are converted into systems of simultaneous nonlinear algebraic equations, which are solved by an iterative process using linear approximations to nonlinear equations at consecutive iteration points.

Let the solution of a nonlinear system $F(X) = 0$ be denoted by X^*. The Newton-Raphson iteration solves the original system of nonlinear equations through a sequence of linear approximations to the nonlinear function $F(X)$ at points $X^{(j)}$, $j = 1, 2, \ldots$

$$F(X^{(j)}) + G(X^{(j)})(X^* - X^{(j)}) \approx 0$$

where G is the Jacobian of F with respect to X (evaluated at $X^{(j)}$). The $(j+1)$ approximation to the solution X^* is obtained by solving a system of linear equations with respect to the correction $\Delta^{(j)}$

$$G(X^{(j)})\Delta^{(j)} = -F(X^{(j)})$$

and $X^{(j+1)} = X^{(j)} + \Delta^{(j)}$. The iteration terminates when $\Delta^{(j)}$ is sufficiently small. The linearized system is solved by LU decomposition and forward/backward substitu-

tion. Efficient solutions of linear systems can improve the performance of the Newton-Raphson iteration. Therefore, we focus on different approaches to linear solutions in this chapter.

Throughout the literature, linear systems are commonly written in the following form:

$$Ax = b,$$

where A denotes the coefficient matrix of the system, b is the vector of the right-hand sides, and x is the vector of unknowns. It is usually assumed that A is a nonsingular $n \times n$ matrix as follows:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

The approaches to solve linear equations can be generally grouped in two categories, direct methods and iterative methods. In the following sections, we will review the background of these techniques for solving a large sparse linear system of equations. Previous work on applying these techniques of parallel/distributed architectures is also discussed.

3.1 Direct and iterative solutions of linear systems

Direct methods of solving linear systems generally use a decomposition of the coefficient matrix or Gaussian elimination. The common form of Gaussian elimination subtracts multiples of rows of A from other rows in order to reduce the matrix to an upper triangular system, which is then solved by back substitution. LU decomposition and Choleski decomposition (for symmetric, positive definite linear systems) are two of

the most commonly used approaches. In LU decomposition, the coefficient matrix is factorized into $A = LU$, where L is lower triangular with 1's on the main diagonal and U is upper triangular. Then the solution is obtained by solving the triangular systems

$$Ly = b,\ Ux = y,$$

which are called the forward and backward substitutions.

Since the matrices generated in circuit simulation are very sparse, we are concerned about direct methods for solving a sparse system of linear equations. A matrix is sparse if many of its coefficients are zero. For practical reasons, it is not necessary to exploit all the zeros. Term **entry** is used to refer to coefficients that are handled explicitly. All nonzeros are entries and some zero coefficients may also be entries.

The exploitation of sparsity can lead to enormous computational savings. The solution of a sparse system is usually divided into several phases [7]:

1. Analysis of the sparsity structure to determine a pivot ordering.
2. Symbolic factorization to generate a structure for the factors.
3. Numerical factorization.
4. Solution of equations.

When it is important to consider numerical values in choosing the pivots, the first three phases are combined into the analyze-factorize phase. The algorithms for direct sparse linear solvers are grouped into three categories: general techniques, frontal methods, and multifrontal approaches. The multifrontal approach is an extension of the frontal method. Details of frontal methods and multifrontal methods are described in [8].

The main features of the general approach, typified by Harwell Subroutine MA28 or Y12M [33], are that numerical and sparsity pivoting are performed at the same time. The major concern is that the factors L and U will be denser than the original matrix A. Fill-in is caused by the operation of Gaussian elimination

$$a_{ij} := a_{ij} - a_{ik}a_{kk}^{-1}a_{kj},$$

when the original value of a_{ij} is zero. So the ordering of A is very important to preserve the sparsity in the factors. Sparsity pivoting is used to find a reordering of the matrix such that the number of operations and fill-in of the reordered matrix are small. The Markowitz strategy [7] is simple and effective for maintaining sparsity for general-purpose use. At each step of Gaussian elimination, the pivot is selected as the nonzero entry of the remaining reduced submatrix with the lowest product of the number of other entries in its row and the number of other entries in its column. For example, after the kth step of Gaussian elimination, let row_i^k denote the number of entries in row i of the reduced $(n-k) \times (n-k)$ submatrix, and col_j^k, the number of entries in column j. The Markowitz strategy chooses such entry $a_{ij}^k \neq 0$ as the pivot which minimizes the expression

$$(row_i^k - 1)(col_j^k - 1).$$

The parallelism of general sparse direct methods will be discussed in the next section.

Iterative methods are attractive for use on vector or parallel computers. Generally, an iterative method is suitable only for a specific class of problems, since the rate of convergence depends on spectral properties of the matrix. No single iterative method is robust enough to solve all sparse linear systems accurately and efficiently. Many iterative approaches are based upon the following approach. The matrix A is split into

an easily invertible part P and a remainder Q. The splitting $A = P - Q$ leads to the basic iteration

$$Px_{i+1} = Qx_i + b, i = 0, 1, 2, \ldots,$$

where x_0 is a user-specified starting vector. If we write $\gamma_0 = b - Ax_0$, then it follows by induction that x_i can be expressed as

$$x_i = x_0 + \alpha_0 P^{-1}\gamma_0 + \alpha_1 P^{-1}AP^{-1}\gamma_0 + \ldots + \alpha_i (P^{-1}A)^{i-1}P^{-1}\gamma_0$$

Consequently, x_{i+1} is equal to x_0 plus a specific vector from i-dimensional subspace spanned by the vectors $P^{-1}\gamma_0, P^{-1}AP^{-1}\gamma_0, \ldots, (P^{-1}A)^{i-1}P^{-1}\gamma_0$. Such a space of the form

$$\text{span}\{f, Bf, B^2 f, \ldots, B^{i-1}f\}$$

is called the i-dimensional Krylov subspace corresponding to f and B, and is denoted as $K_i(B; f)$. In our case we have $x_i = x_0 + y$, with $y \in K_i(P^{-1}A; P^{-1}\gamma_0)$.

Often P is called the preconditioner for the system $Ax = b$. Note that the special choice $P = I$ leads to solution elements belonging to $K_i(A; \gamma_0)$, which gives rise to the unpreconditioned or basic methods, such as Jacobi's method and Seidel's method.

The straightforward iteration leads to very special elements of the Krylov subspace. But we can also search for more optimal elements. Such an approach leads to the so-called Krylov subspace methods or projection-type methods such as BG (the biconjugate gradient method), CGS (the conjugate gradient-squared method), GMRES, and many more.

Hybrid solutions combine the capabilities of direct methods and iterative methods. When solving a sequence of linear systems, instead of factoring each linear system to be

solved, the iterative method is used whenever appropriate. This approach was shown to be very efficient by Liegmann in [17].

3.2 Parallel direct linear solvers

In SPICE-PAC, subroutines SPPDCD and SPPDCS solve the linear equations. They use the LU decomposition (SPPDCD) followed by forward/backward substitutions (SPPDCS). The coefficient matrix is assumed to be nonsingular, sparse and general. The next two sections discuss different approaches to LU decomposition used for shared-memory and distributed-memory architectures.

3.2.1 Shared-memory architectures

In shared-memory architectures, all processors are connected with (shared) memory modules by an interconnection network. Memory access resolution schemes are needed to handle the situations when two (or more) processors attempt to access the same memory module at the same time. Optimal performance on shared-memory computers requires algorithms that minimize data movement between the shared memory and processors. Figure 3.1 is a very simple outline of a shared-memory architecture; processors are indicated by P–blocks, and memory modules by M–blocks. Each processor can access any of the memory modules, and processors exchange information by using common (the same address space) memory locations.

The sparsity of the matrix can be used to exploit parallelism in parallel LU decomposition of general unsymmetric sparse matrices. The idea is to use the ability to choose several pivots simultaneously. Two matrix entries, a_{ij} and a_{rs}, can be used as pivots simultaneously if a_{is} and a_{rj} are zero. These pivots are called *compatible*. This

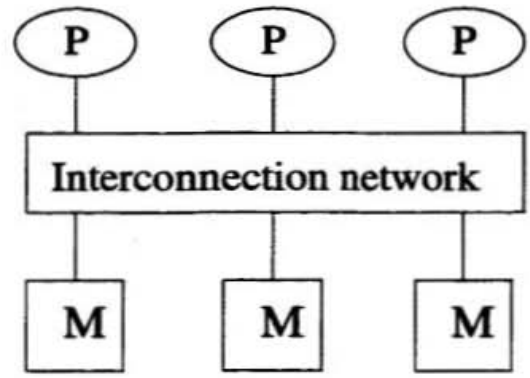


Figure 3.1: Shared-memory architecture.

method has been applied to several parallel algorithms for general matrices [3]. The main process is to select a number of compatible pivots that would create a diagonal block if ordered to the top left of the matrix. The update process from all these pivots is then performed in parallel. This procedure is then repeated on the reduced matrix. Different algorithms may select the pivots and update the matrix differently. However, any one of them must compromise the Markowitz criterion to get a large compatible pivot set.

An incompatible table [3] is used to assist in the pivot search. The two-stage implementation chooses pivots in parallel from the diagonal and then off-diagonal pivots are chosen sequentially to stabilize the ordering. Thresholds for both sparsity and stability are set when choosing pivots. Pivoting for numerical stability is performed in a different section of the code. Once the set of compatible pivots is selected, each pivot is tested for numerical stability.

The decomposition of the coefficient matrix is followed by forward and backward substitutions. Based on the block triangular structure of L and U, two methods are tested in [3], a block approach and an asynchronous approach. In the block approach, the matrix is divided into regions and the rows of each region are processed in parallel

by a prescheduled parallel loop. The solution of the next region is not started until the previous region is complete. The asynchronous approach is a self scheduled process and it processes rows of the matrix in forward or backward substitutions. However, the processing of the next region is started without the requirement of completion of the previous region. Thus access to the elements in the unknown vectors should be synchronized. The performance comparison of the two methods showed that the asynchronous approach has much higher execution speed when the block sizes and the number of parallel processors are increased [3]. For a 4-processor parallel environment, the performances of the two methods are almost the same.

The most commonly used codes for solving general sparse linear systems are MA28 and Y12M. They are two sets of Fortran subroutines for sparse unsymmetric linear equations.

MA28, developed by I.S. Duff, is part of the Harwell Subroutine Library. The user can set a parameter u to control the balance between numerical pivoting and sparsity pivoting. $u = 1.0$ gives partial numerical pivoting, while $u = 0.0$ minimizes fill-in without checking the magnitude of the pivots. The sparsity pivoting is based on Markowitz criterion. Since MA28 performs LU decomposition followed by forward/backward substitution in separate subroutines, it is suitable for solving a sequence of linear systems with the same structure by performing one decomposition.

Y12M was developed by Zlatev for the same purpose of MA28 [33] (the code is available at netlib *http://www.netlib.org*). The Y12M algorithm extends the notion of compatible pivots by permitting the pivot block to be upper triangular rather than diagonal, which allows selection of a larger number of pivots. However, in this case, the update is more complicated. The code selects the mixture between sparsity and numer-

ical pivoting itself. The underlying sparsity pivoting strategy is based on Markowitz criterion as well.

Y12M and MA28 were tested [29] on a uniprocessor machine and a Cray C98/4256, a shared memory computer with four processors (in the test, on average 2.5 processors were used concurrently). A tool called ATExpert was used to predict the speedup factors on four processors. The result confirms the expectation that the speedup factors grow for increasing problem size.

PARASPAR (A Package for the Solution of Large and Sparse Systems of Linear Algebraic Equations on Parallel Computers with Shared Memory) is a another set of Fortran subroutines for solving systems of linear equations whose coefficient matrices are assumed to be generally sparse [32]. Four methods are available in this package: direct solution by Gaussian elimination, iterative refinement, preconditioned ORTHOMIN algorithm and the pure ORTHOMIN algorithm. Two different single pivoting strategies and a parallel pivoting strategy are provided. If the matrix is very sparse and remains sparse during the computations, the parallel pivoting is recommended.

3.2.2 Distributed-memory architectures

The other class of parallel computers uses memory which is distributed among the processors; each processor contains a CPU and local memory. Distributed memory can be shared or not. Distributed-shared-memory architectures, or hybrid architectures, are becoming quite popular recently. If the memory is not shared, processors use messages to communicate and exchange information, and the systems are called message-passing architectures. Figure 3.2 outlines a distributed-memory architecture (M' denotes local memory) and a hybrid architecture.

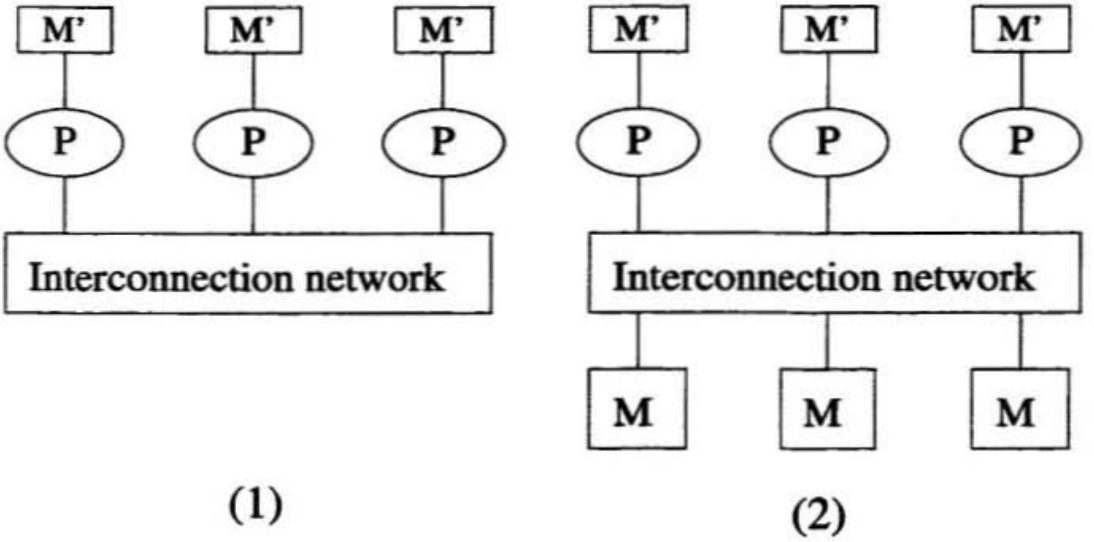


Figure 3.2: Distributed-memory architecture (1) and hybrid architecture (2).

The pattern of connections between processors is called the topology of the parallel computer. The most common topologies are ring, mesh (usually two or three dimensional) and n-cube. The communication overhead is critical in distributed computing. An important issue in developing algorithms for distributed-memory architectures is how to reduce the data communication between processors. The extreme case of distributed-memory architecture is a network of computers, a concept very fashionable recently.

Sadayappan and Rao [26] analyzed the amount of communication in sparse LU decomposition on a distributed-memory parallel computer. They present the fragmented distribution which splits rows and columns into parts and distributes these parts over different processors. This is in contrast to the shared-memory algorithms that treat rows and columns as basic indivisible units. Compared to a row/column-wrapped distribution, the fragmented distribution decreases the total length of communication messages.

Skjellum [27] presents an algorithm using the grid distribution defined below:

$$A_{ij} \longmapsto \text{processor } (i \bmod Q, j \bmod Q) \text{ for all } i, j,\ 0 \leq i, j < n,$$

for Q^2 processors (s,t), $0 \leq s,t < Q$. This distribution splits each row i into Q row parts, i.e., sets of the form $\{A_{ij} : 0 \leq j < n \wedge j \bmod Q = t\}$, and it also splits each column into Q column parts.

Experimental results on a Symult s2010 show that for a 2500 × 2500 random sparse matrix with approximately 51 nonzeros per row, execution time on a 96 processor machine is 9.7 times faster than on a 6 processor machine.

Stappen, Bisseling and Vorst [28] developed an algorithm based on an approach used for dense matrices. The same grid distribution as above was used. The algorithm scales well with the number of processors, and it achieves a speedup of up to 107 on 400 processors for large problems. This algorithm is suitable for transputer networks and hypercubes.

ScaLAPACK is a portable linear algebra library for distributed memory computers. The ScaLAPACK library includes a subset of LAPACK routines redesigned for distributed memory MIMD parallel computers. It is written in an SPMD style using message passing for communication. It assumes that matrices are laid out in a two-dimensional block cyclic decomposition. The fundamental building blocks of the ScaLAPACK are distributed memory versions (PBLAS) of the Level 1, 2 and 3 BLAS (Basic Linear Algebra Subroutines), and BLACS (Basic Linear Algebra Communication Subprograms) for communication tasks.

PBLAS is very similar to BLAS. Only one substantially new routine, matrix transposition, was added to PBLAS, since it is a complicated operation in a distributed memory system [35]. As BLAS provides a shared memory standard, PBLAS will provide a standard for distributed memory systems. So, PBLAS is supposed to simplify

and stimulate the development of high performance and portable parallel numerical software.

The model corresponding to the parallel LU factorization implemented in ScaLAPACK is presented in [35]. Its performance was measured for different block sizes and grid sizes. Most of the computations of the ScaLAPACK routines are performed in a block fashion using Level 3 BLAS. The computational blocking factor is chosen to be the same as the distribution block size. The performance of the ScaLAPACK library is not very sensitive to the block size, as long as the block sizes are similar. The best grid shape is determined by the algorithm implemented in the library and the underlying physical network. The LU factorization performs better for process grids when the number of processors in rows is smaller than that in columns, which can be denoted as $P_r < P_c$.

ScaLAPACK is portable across the Intel series(NX), IBM SP series, Thinking Machines CM-5, and clusters of workstations via PVM and MPI. Implementations of ScaLAPACK on networks of workstations were conducted over general sparse matrices. The performance on 2, 4, and 8 workstations is poor with almost no speedup. The problem is due to frequent data transmission between workstations and low network speed. ScaLAPACK is more suitable for closely coupled distributed-memory computers. Some specialized algorithms should be used for solving large sparse linear systems on workstation clusters.

3.2.3 Clusters of workstations

While dedicated distributed-memory systems are expensive and many users do not have access to them, workstation clusters are quite popular and relatively inexpensive.

Bjorstad, Coughran and Grosse [4] applied domain decomposition techniques to modeling semiconductor devices on a network of HP workstations connected by FDDI. FDDI (Fiber Distributed Data Interface) is a high-performance fiber optic token ring LAN, which can accommodate up to 1000 stations on a cable of up to 200km communicating at up to 100 Mbps.

"Domain decomposition" refers to a method that divides the original problem domain into parts and solves each subdomain locally. The results show that domain decomposition is one of those "essentially independent parallel computations" that works well for a modest number of workstations. The speedup they gained on four workstations was 2.9.

Peter Carlin [5] modified some basic linear algebra algorithms to improve their performance on networks of workstations. This library was designed with consideration towards three characteristics of networks of workstations: small numbers of processes, availability of multithreading and high communication latency.

A domain decomposition approach is a natural way to coarse grain parallel computation: it decomposes the sparse coefficient matrix of linear equations into blocks and has each block solved on a different processor. When partial solutions are collected from each processor, the final result can be assembled into the solution vector. There is no existing code for solving sparse linear systems by domain decomposition. The next chapter describes the design and development of this approach on a network of workstations.

3.3 Parallel iterative solvers

Iterative methods are very different from direct methods. The idea of an iterative solution is to use a predicted initial approximation as a starting-point solution of the linear system, and to refine it iteratively until the final solution is reached. So, when the starting point is "close" to the solution, the iterative solver can be very efficient. However, convergence can be a problem. If a sequence of "similar" linear equations is to be solved (as is the case for solving systems of nonlinear equations), the iterative approach may be very attractive, because the solution for one system can be used as the starting-point for the next system of equations.

In the beginning of this chapter, the standard Krylov subspace acceleration was introduced. The main operations in this method are: (1) vector update, (2) dot product, (3) matrix-vector product and (4) preconditioning. The most expensive operations in a parallel iterative solver are the matrix-vector product and preconditioning.

Lo and Saad [22] proposed a parallel iterative approach by using domain decomposition. Each processor holds a set of equations (rows of the linear system) and a vector of the variables associated with these rows. Then each system associated with a subdomain is solved by an iterative process. We need to multiply the matrix consisting of rows that are local to a given processor by a distributed vector. Some components of the vector are local (called local variables), while other components (called external variables) have to be transferred from other processors.

Let A_{loc} be the local matrix, i.e., the rectangular matrix consisting of all the rows of A that are allocated to the given processor. B_{loc} is the submatrix of A_{loc} whose nonzero elements bl_{ij} are such that j is a local variable. B_{loc} is a square matrix of size $n_{loc} \times n_{loc}$ where n_{loc} is the number of equations assigned to this processor. Similarly, B_{ext} is the

submatrix of A_{loc} whose nonzero elements be_{ij} are such that j is not a local variable. The following steps are needed to perform a matrix-vector product:

1. Multiply the diagonal block B_{loc} by the local variables.

2. Bring in the external variables.

3. Multiply the B_{ext} by these external variables and add the result to the result of step 1.

Steps 1 and 2 can be performed simultaneously. A processor can be multiplying B_{loc} by the local variables while waiting for the external variables.

On the preconditioning side, Saad developed a flexible variant of the GMRES algorithm (FGMRES) [23]. FGMRES allows the inner preconditioning steps to be completely asynchronous in order to minimize communication and synchronization costs in a parallel approach.

P-SPARSLIB was developed using the above method by Saad and Malevsky [24]. They used Harwell-Boeing test problems. Different network configurations were tested and a SP2 cluster achieved a fairly good performance for the solution of unstructured sparse systems. For a 4-processor SP2 machine, the speedup was 2.92, and for an 8-processor machine, the speedup was 4.62.

3.4 Parallel hybrid solvers

Some work has been done in developing efficient solutions of sparse linear systems using a combination of direct and iterative methods. The idea of so-called hybrid methods is that during the solution of a problem, one can select the method which is known to work best in a particular phase of the solution process, whereas the normal approach

uses only one method for the whole solution process. For example, Liegmann [17] used a combination of both iterative and direct techniques. When solving a sequence of linear systems, the hybrid approach tries to avoid as many computationally expensive factorizations as possible by using an iterative method instead. To increase the likelihood of success of the iterative algorithm, the last factorization is used as a preconditioner. This approach has shown to save up to 90% of the factorizations. Therefore, for a sequence of linear systems with similar sparsity structure, the hybrid method of using the solution from direct methods as the starting point vector of the iterative process can be effective.

Chapter 4

The Domain Decomposition Method

Since dedicated supercomputers are rather expensive, and often a whole team is needed to service a single system, readily available networks of workstations connected either by Ethernet or FDDI offer an attractive alternative. The overall power of such a workstation cluster can often be equivalent to that of a supercomputer. Most of the utilization of workstations is low (e.g. if users are editing files). Therefore it can be very economical to use these machines by running parallel application software on such clusters. However, there is a disadvantage to implementing parallel programs on workstation clusters: the communication overhead is high because the transmission performance of the network is still rather low and is limited by the underlying hardware (e.g. Ethernet: 10MB/s) [30].

High communication overhead of workstation clusters implies a constraint on the choice of parallel algorithms. Only those algorithms which do not require extensive communication overhead can achieve good speedup on workstation clusters. Low-level parallelism, such as that of independent loops, can be successfully applied only on shared memory systems or on massively parallel tightly coupled computers. Thus,

automatically parallelized code based on low level parallelism generally cannot achieve good speedup on workstation clusters.

New algorithms have to be developed for workstations. One approach which is very promising is the domain decomposition method. The idea of this method is to split a problem domain into subdomains: each subdomain is calculated by an individual processor, so the application can run in parallel. Each parallel task consists of two parts, a local part and a part which requires communication with other tasks. In analyzing the time complexity of a parallel program, there are two major costs: one is communication, and the other is load balancing. Problems which can be partitioned in such a way that the subtasks do not need to communicate are known as perfectly parallel problems. Application programs which, for example, involve the solution of PDEs (partial differential equations) do not fall into this class of problems [14]. Besides reducing the communication overhead to improve performance, the workload of parallel processors must be balanced, otherwise some processors have to wait for others to finish their work before they can proceed with their next task. In [14], Gropp gave an example of a system where $p-1$ processors have work load W_1 and the last one has work load $W_2 > W_1$. For such a case, the best possible speedup is :

$$S_p = \frac{(p-1)W_1 + W_2}{W_2} = 1 + (p-1)\frac{W_1}{W_2}. \tag{4.1}$$

Formula 4.1 shows that smaller speedup is expected if there is a large imbalance in workload distribution. Figure 4.1 shows S_p with varying p and $\frac{W_1}{W_2}$.

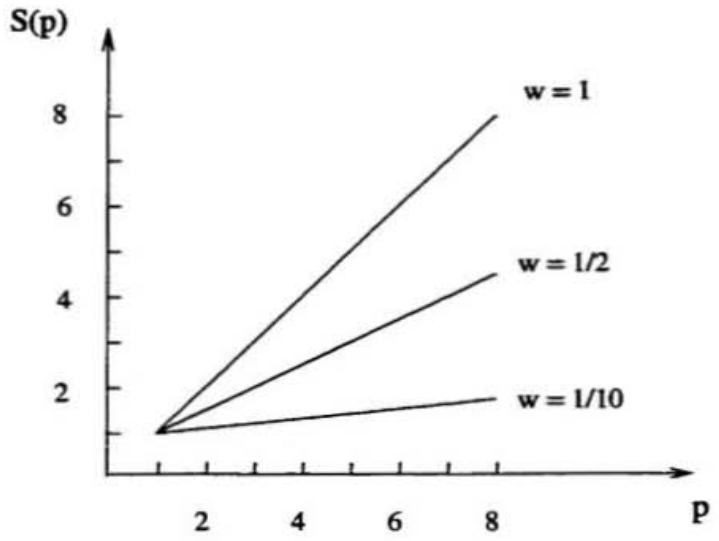


Figure 4.1: Speedup of a p processor system with different workload distributions W.

4.1 An overview of the domain decomposition approach

To solve sparse linear systems by domain decomposition, it is natural for us to think about partitioning the coefficient matrix into submatrices, and solving each subsystem independently of others. The partition method for a structurally symmetric matrix has been developed and known for many years. Liegmann [17] applied this method in his experiment of solving linear equations on a network of workstations.

Using domain decomposition requires a reordering of the coefficient matrix A into a block diagonal form which is called the "arrowhead structure":

$$\begin{pmatrix} A_1 & & & M_1 \\ & \ddots & & \vdots \\ & & A_n & M_n \\ N_1 & \dots & N_n & A_s \end{pmatrix}$$

The diagonal block A_s in the lower right corner is referred to as the **separator** block. The ordering is achieved by an algorithm which is based on graph theory and which is

described in Section 4.3.1.

Once the matrix is reordered, the linear system $Ax = b$ is multiplied by the matrix

$$\begin{pmatrix} A_1^{-1} & & & \\ & \ddots & & \\ & & A_n^{-1} & \\ & & & I_s \end{pmatrix}$$

which results in:

$$\begin{pmatrix} I_1 & & & A_1^{-1}M_1 \\ & \ddots & & \vdots \\ & & I_n & A_n^{-1}M_n \\ N_1 & \dots & N_n & A_s \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ x_s \end{pmatrix} = \begin{pmatrix} A_1^{-1}b_1 \\ \vdots \\ A_n^{-1}b_n \\ b_s \end{pmatrix}.$$

Let $\tilde{M}_i = A_i^{-1}M_i$ and $\tilde{b}_i = A_i^{-1}b_i$. The above equation can be written as a set of $n+1$ blocks of linear equations expressed as

$$x_i + \tilde{M}_i x_s = \tilde{b}_i, \; i = 1, \dots, n;$$

$$\sum_{i=1}^{n} N_i x_i + A_s x_s = b_s.$$

Therefore, $x_i = \tilde{b}_i - \tilde{M}_i x_s$, for $i = 1, \dots, n$. After substitution, we get

$$(A_s - \sum_{i=1}^{n} N_i \tilde{M}_i) x_s = b_s - \sum_{i=1}^{n} N_i \tilde{b}_i.$$

The term

$$A_s - \sum_{i=1}^{n} N_i \tilde{M}_i$$

is referred to as the **Schur complement** of A_s. Computing the Schur complement of A_s is the major task in the parallel solution process. After x_s is computed, each processor i can compute its local portion x_i of the overall solution vector x.

4.2 The Master-Slave model

The above solution can be used in the Master-Slave model of parallel computation. In this model, the master process controls the other slave processes and computation is performed only by slave processes.

---

partition the coefficient matrix A;
create the slave processes;
broadcast data to slaves;
do
  if *a partial result from a slave has arrived* **then**
    place the partial result into overall solution
  end if
until *all slaves have sent results;*

Algorithm 4.1: The Master Process.

---

receive *data from master;*
extract local data;
compute local part of Schur complement of A_s*;*
do
  receive other parts of A_s
until A_s *is complete;*
solve the local block of linear system;
send *result to master;*

Algorithm 4.2: The Slave Process.

- *The MASTER process*:

  Algorithm 4.1 explains the master process. The master process consists of four steps. First, the coefficient matrix A is reordered into the arrowhead structure. Then the matrix is partitioned into separate blocks, A_i, M_i and N_i. This transformation should also be applied to the right hand side of the linear system. Second,

the slave processes are created. The creation of slave processes can be done by using a communication library like PVM (Parallel Virtual Machine) to start the executable program which resides on each "slave" processor. In the next step, the data structure representing the linear system is broadcast to all slave processes. Each process i extracts the blocks A_i, N_i, M_i and the portion b_i of the right-hand side vector. After broadcasting the data structure to slave processes, the master process waits to receive results from slaves. When a slave process returns its partial solution x_i, the master process will put x_i in the entire solution vector x.

- *The SLAVE process*:

  Algorithm 4.2 describes the slave process which performs the solution of the linear system. Once a slave process is initiated, it waits for data sent from the master process. Depending on its task identifier i, the slave process extracts the appropriate blocks A_i, N_i, M_i and A_s^i of the separator A_s. Then slave processes compute the Schur complement of A_s in parallel, and solve the local block linear system. Eventually, each slave task sends its partial result, x_i, to the master until the final solution vector is complete.

4.3 Matrix partitioning

Direct methods of solving linear system $Ax = b$ usually decompose the coefficient matrix A into lower and upper triangular matrices:

$$A = LU \tag{4.2}$$

This decomposition simplifies the solution, as only forward and backward substitutions are needed for the final solution. The decomposition of A into two triangular factors,

L and U, is called an LU decomposition or LU factorization [6]. When A is sparse, the factorization process can introduce additional non-zero elements in LU factors. These elements are called fill-in entries and such additional non-zero elements require more storage space. This is a very important aspect when direct methods are considered. Moreover, the number of required arithmetic operations increases with the size of the factors.

Reordering the matrix is a natural way to reduce fill-in. Interchanging columns and rows can be done by multiplying the matrix A by permutation matrices P and Q. P interchanges the rows of A and Q permutes the columns. For structurally symmetric matrices, usually $Q = P^T$ is used to preserve the symmetric structure of the reordered matrix. Reordered matrices with no fill-ins in their LU factors are called **perfect elimination matrices.**

Some properties of elimination matrices are [17]:

- A matrix has a perfect elimination ordering if its graph is triangulated.

  A graph G is triangulated if for each cycle $c = (i, i+1, \ldots, i+n, i)$ in G with $n \geq 3$ there is an edge between any two non-consecutive nodes of c. Most matrices associated with real problems do not have triangulated graph representations. Therefore other approaches have been used to retain the sparsity of the coefficient matrices.

- Computing the minimum fill-in is an NP-complete problem.

  There are several alternative approaches producing near optimal reordering for general matrices. Among these approaches, the minimum degree reordering algorithm has proven to be most effective [11]. A detailed description of the minimum

degree reordering is shown in the following subsections.

4.3.1 Graph representation of sparse matrices

Sparse matrices and graph theory are closely linked. The pattern of a square sparse matrix can be represented by a graph. In this chapter, graph theory is used for visualization of sparse matrix partitioning.

A directed graph (digraph) consists of a set of nodes (vertices) and direct edges between nodes. Any square sparse matrix pattern has an associated digraph. For a given sparse matrix A, a node is associated with each row. For each entry a_{ij}, there is an edge from node i to node j in the directed graph as shown in Figure 4.2.

Figure 4.2: An unsymmetric matrix and its directed graph.

For a symmetric matrix a connection from node i to node j implies that there is also a connection from node j to node i. So the edge directions may be dropped. The graph without arrows is called an undirected graph, as illustrated in Figure 4.3.

Figure 4.3: A symmetric matrix and its directed graph.

A special case occurs when an undirected graph contains no cycles. If the graph is

connected and a particular node is selected as the root, it is a rooted tree, as illustrated in Figure 4.4.

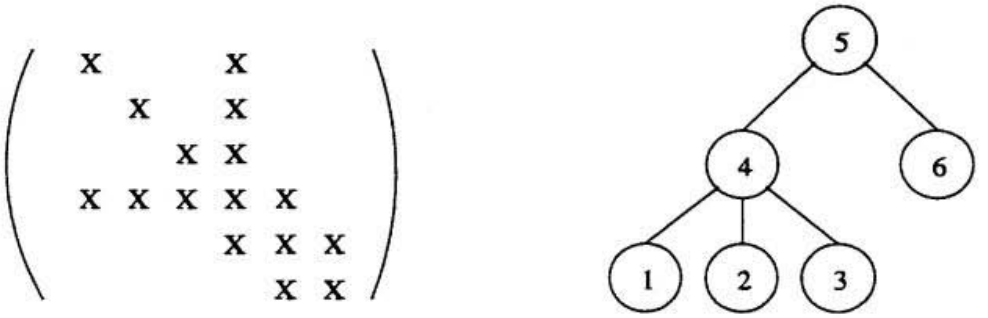


Figure 4.4: A matrix and its rooted tree.

Since the matrix reordering and partitioning algorithms are described using graph theory, some basic terms which are used frequently in the next section are introduced here.

$G = (V, E)$ represents a graph. V denotes the set of nodes (or vertices) and E is the set of edges connecting the nodes of the graph. The elements of E are commonly expressed as node pairs, i.e.

$$E \subseteq \{(i, j) \mid i, j \in V, i \neq j\}.$$

The number of elements of a set A is denoted by $|A|$. Two nodes $i, j \in V$ are adjacent to each other, if $(i, j) \in E$. The adjacency set of a node i is defined as

$$adj(i) = \{j \in V \mid (i, j) \in E\}.$$

Graph $G = (V, E)$ is connected, if for every $i, j \in V$ there is a path p from i to j, i.e.

$$\forall i, j \in V, \exists p = (i, \ldots, j).$$

Graph theory helps in visualizing the changing pattern of entries as elimination takes place. Corresponding to the graph G, the elimination graph G_j for node j is obtained

by removing node j and adding a new edge (i,k) whenever (i,j) and (j,k) are edges of G but (i,k) is not. For example, G_1 for the graph of Figure 4.3 has the representation shown in Figure 4.5, with the new edge $(2,4)$ added.

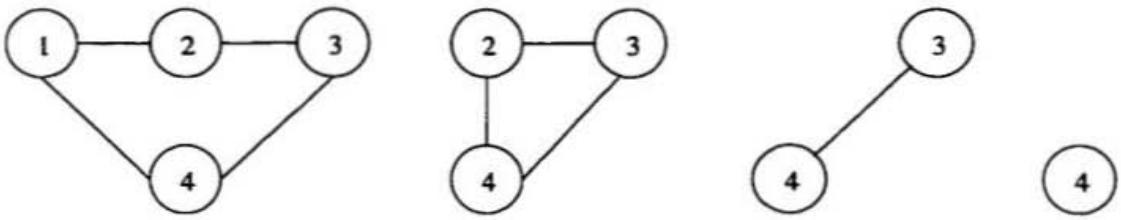


Figure 4.5: Elimination sequence for the matrix from Figure 4.3.

It is important to note that the graph structure of a structurally symmetric matrix does not change when its rows and columns are reordered with the same permutation matrix, i.e. $G(A) = G(PAP^T)$. Changing the permutation matrix just permutes the node numbering in the corresponding graph. Consequently, finding a matrix permutation which generates minimal fill-in is equivalent to finding such a permutation of the nodes which minimizes the size of the adjacent sets of nodes:

$$|adj(i_m)| = \min(|adj(i_n)|),\ i_n \in V \setminus \{i_1, \ldots, i_{m-1}\}$$

for each elimination step m.

4.3.2 The minimum degree algorithm

It is known [6] that sparse matrix factorization requires a reordering of the rows and columns in order to reduce the number of fill-ins. Among various algorithms, the minimum degree approach has been proven to be very effective [11]. Let G be an undirected graph and v a node in G. We use the notion $adj_G(v)$ to refer to the set of nodes adjacent to v in G. The degree of v in G is denoted by $|adj_G(v)|$.

The basic minimum degree ordering can be best described in terms of elimination graphs. We use G_v to represent the elimination graph obtained after the elimination of v from the graph G. The graph G_v can be obtained by deleting v and its incident edges from G and then adding edges to connect the nodes that were adjacent to v into a clique (a clique is a subgraph where every two nodes are adjacent). If v is not adjacent to y in G, then:

$$adj_{G_v}(v) = adj_G(v).$$

If $v \in adj_G(y)$, we have

$$adj_{G_y}(v) = (adj_G(y) \cup adj_G(v)) - \{v, y\}.$$

The basic algorithm is described as follows:

$S := \{\}$;
G is the graph of the matrix;
while $S \neq V$ **do**
  for $v \in V \setminus S$ **do**
    $\delta(v) = |adj(v)|$
  end for;
  select $z \in \{y \in V \setminus S \mid \delta(y) = \min_{v \in V \setminus S}(\delta(v))\}$;
  $S := S \cup \{z\}$;
  eliminate z and create G_z
end while;

Algorithm 4.2.1: The minimum degree algorithm.

Once z with minimum degree is selected, it is added to S, the set of reordered nodes, and is eliminated from the current elimination graph. Rules on how to select z are known as tie-breaking strategies. Effective tie-breaking can significantly reduce the number of fill-in entries. Most implementations of reordering use random tie-breaking,

which is to select the nodes randomly, since there is no efficient tie-breaking procedure available for general sparse linear systems.

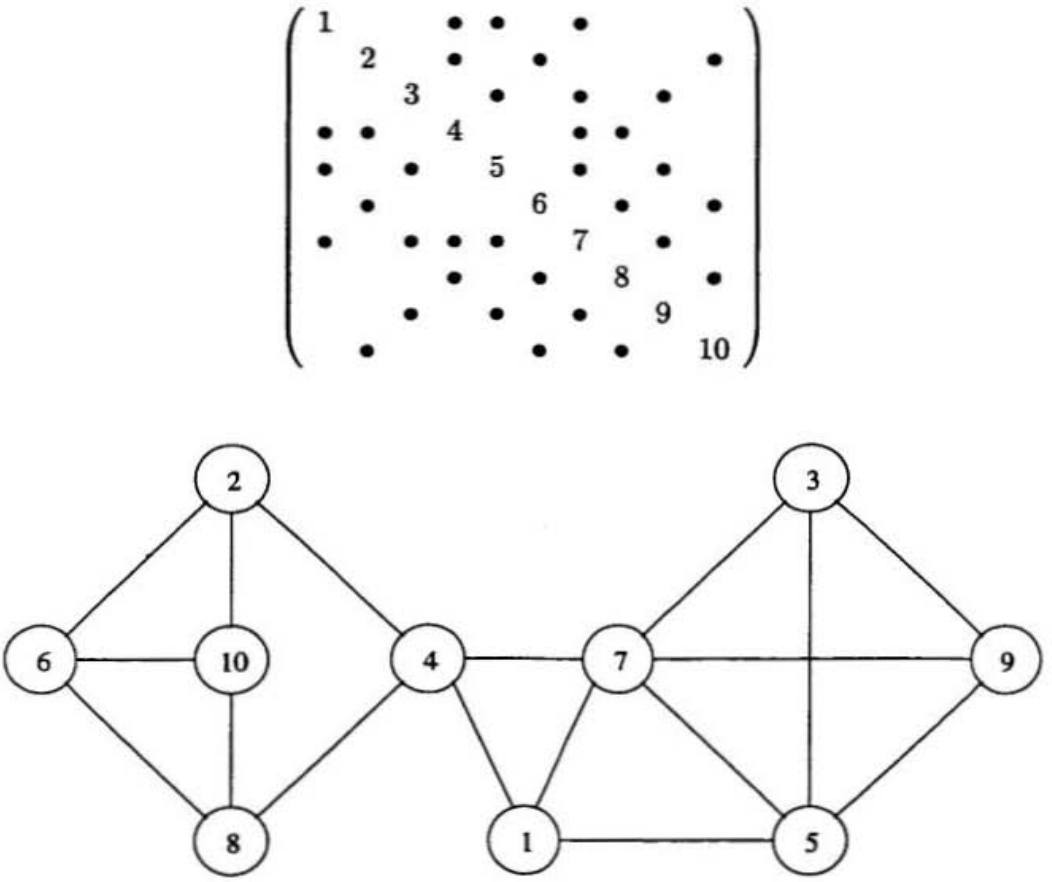


Figure 4.6: A structurally symmetric matrix pattern and its graph representation.

Here is an example. A structurally symmetric matrix and its graph representation are given in Figure 4.6. The matrix will be reordered by applying the minimum degree algorithm.

The initial minimum degree is $\delta_0 = 3$. We have

$$S = \{\}, T = \{1, 2, 3, 6, 8, 9, 10\}, \delta_0 = 3.$$

Since random tie-breaking is used, any node in T can be chosen for elimination. If we

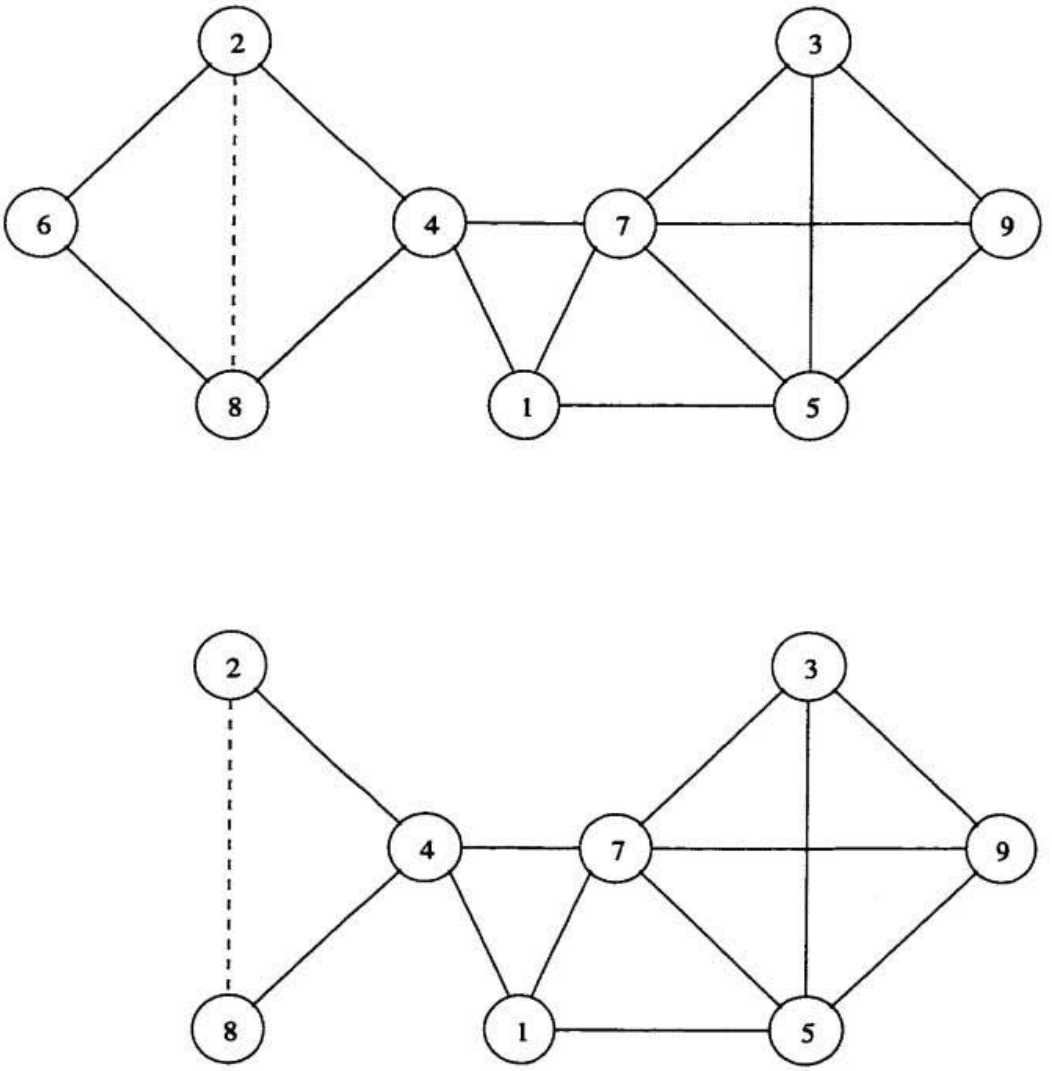


Figure 4.7: Elimination graphs after removing node 10 and node 6.

set $z = 10$, the elimination graph G_z turns out to be like that in Figure 4.7. The dash line indicates the fill-in edge. The minimum degree δ_1 decreases to 2. In the next loop we have:

$$S = \{10\}, T = \{6\}, \delta_1 = 2.$$

Apparently we can only set $z = 6$. The elimination of node 6 does not introduce any new edge because the nodes form a clique with its adjacent nodes in the current graph. This is true for the remaining steps of eliminations. Finally, we have the permutation $P_1 = (10, 6, 2, 8, 4, 1, 5, 3, 7, 9)$. The filled matrix pattern is shown in Figure 4.8. The permutation P_1 results in an optimal elimination sequence with only two fill-in entries.

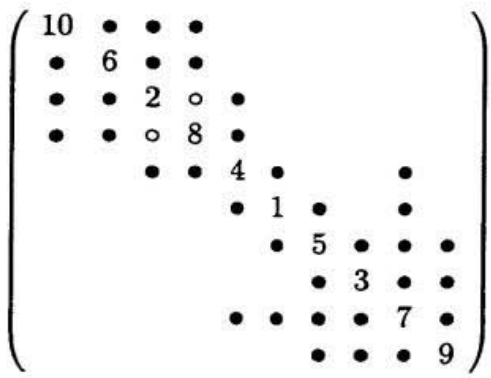


Figure 4.8: Pattern of the filled matrix F of $P_1 A P_1^T$.

For the same matrix, if we select $z = 1$ in the first step instead of $z = 10$, we will have:

$$S = \{1\}, T = \{2, 3, 6, 8, 9, 10\}, \delta_1 = 3.$$

Elimination of node 1 introduces one fill-in edge. With respect to the number of fill-in edges, there is no difference between eliminating node 1 or node 10 in the first step.

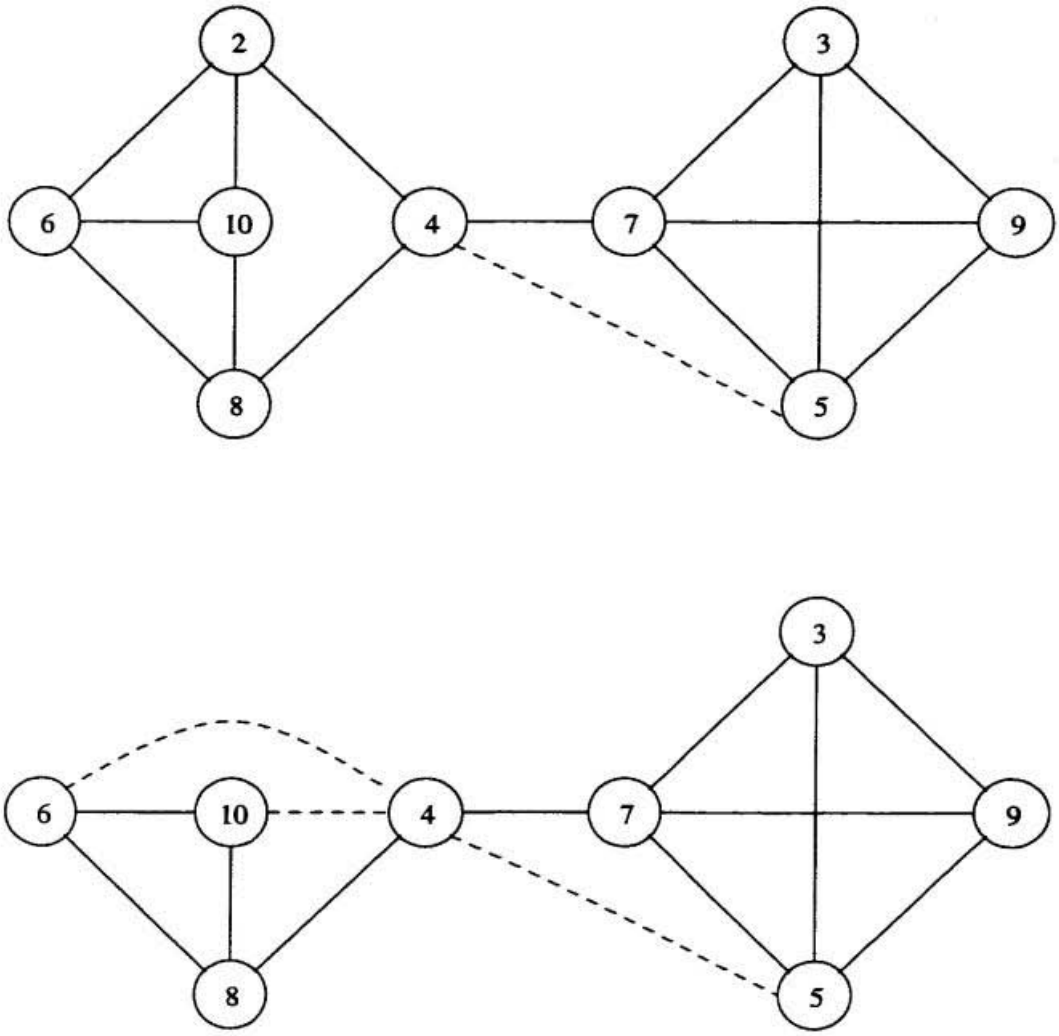


Figure 4.9: Elimination graphs after removing node 1 and node 2.

However, the situation changes if we select node 2 to be eliminated next. Removing node 2 results in two additional fill-in edges (4, 10) and (4, 6), as shown in Figure 4.9. Elimination of the remaining nodes does not generate fill-ins. The alternative permutation is $P_2 = (1, 2, 6, 8, 10, 4, 7, 3, 5, 9)$. The filled matrix $F_{P_2AP_2^T}$ for permutation P_2 is shown in Figure 4.10. From this picture we can see that $P_2AP_2^T$ has four additional fill-in entries compared to the filled matrix $P_1AP_1^T$.

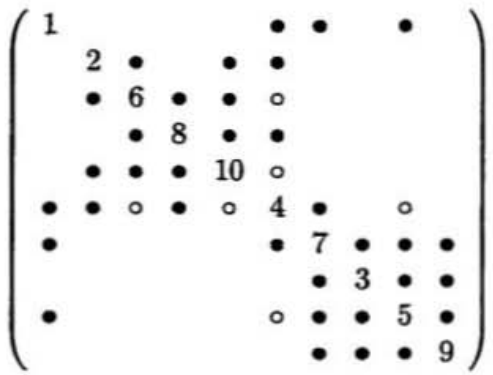


Figure 4.10: Pattern of the filled matrix F of $P_2AP_2^T$.

4.3.3 Improvements to the minimum degree algorithm

The transformation of the elimination graph G_y is the central part of the minimum degree algorithm. According to the Algorithm 4.2.1, the degrees of nodes adjacent to node y may change. So a re-calculation of their degrees is needed for the next node selection. It has been recognized that this "degree update" is the most time consuming part of the entire algorithm. George and McIntyre [12] observed that in the elimination of the minimum degree node y, usually there is a set of nodes adjacent to y that can be eliminated right after y.

Theorem 5.1 [11]: If y is selected as the minimum degree node in the graph G, then

$$Y(y) = \{z \in adj_G(y) \mid degree_{G_y}(z) = degree_G(y) - 1\}$$

can be selected next (in any order) in the minimum degree algorithm.

In other words, instead of performing a degree update after the elimination of node y, we can simultaneously eliminate the nodes in $Y \cup \{y\}$. Thus, we can save the computation time in the degree update steps. To identify Y, we introduce the definition

of **indistinguishable nodes.**

Definition: In graph $G = (V, E)$, y and z are indistinguishable if $adj_G(y) \cup \{y\} = adj_G(z) \cup \{z\}$.

Theorem 5.2 [11]: Let $z \in adj_G(y)$. $degree_{G_y}(z) = degree_G(y) - 1$ if and only if $adj_G(y) \cup \{y\} = adj_G(z) \cup \{z\}$.

It is obvious that if two nodes are indistinguishable, their degrees must be the same, so they can be merged and treated as one node. Theorems 5.1 and 5.2 imply that indistinguishable nodes can be eliminated together whenever one of them is picked for elimination. In this way, we need only consider one representative node for a group of indistinguishable nodes. In the literature, these representative nodes have been referred to as **supernodes.** The simultaneous elimination of indistinguishable nodes during minimum degree reordering is called **mass elimination.** The minimum degree algorithm modified for mass elimination is shown as Algorithm 4.3.1.

$S := \{\}$;
while $S \neq V$ **do**
  for $x \in V \setminus S$ **do**
    $\delta(x) = |adj(x)|$
  end for;
  pick $z \in \{y \in V \setminus S \mid \delta(y) = \min_{x \in V \setminus S} \delta(x)\}$;
  $Y := \{y \in V \setminus S \mid y$ *indistinguishable from* $z\}$;
  $S := S \cup Y$;
  eliminate Y *and determine the new graph*
end while;

Algorithm 4.3.1: The minimum degree algorithm with mass elimination.

The use of indistinguishable nodes not only saves elimination time but also the time of degree updates. Another enhancement of the minimum degree algorithm is proposed in [18]. Instead of performing the degree update step after each minimum degree node

selection and elimination graph transformation, the technique of multiple elimination postpones the degree update step to a later stage. In the elimination of node y from the graph G, the structure associated with nodes not in $adj_G(y)$ remains unchanged. The idea is to suspend the degree update for nodes in $adj_G(y)$ and select a node with the same degree as y in the remaining subgraph $G - (adj_G(y) \cup \{y\})$. This process is repeated until there are no nodes in the remaining subgraph with degree equal to that of y. A degree update is then performed. In essence, before each degree update step, an independent set of nodes with minimum degree is selected. The fact that the set is independent allows the degree update to be delayed until the entire set is determined and eliminated. The minimum degree algorithm with multiple mass elimination is referred to as the multiple minimum degree algorithm. Algorithm 4.3.2 shows the details of the multiple minimum degree algorithm.

---

for $x \in V$ **do**
  $\delta(x) := |adj(x)|$
end for;
$S := \{\}$;
while $S \neq V$ **do**
  $T := \{y \in V \setminus S \mid \delta(y) = \min_{x \in V \setminus S} \delta(x)\}$;
  for $y \in T$ **do**
    if y *is not marked* **then**
      $Y := \{x \in T \mid x$ *indistinguishable from* $y\}$;
      mark all nodes in $adj(Y)$ *and* Y;
      $S := S \cup Y$
    end if
  end for;
  eliminate all marked nodes in S *from the graph*;
  for *all marked nodes* $x \in V \setminus S$ **do**
    $\delta(x) = |adj(x)|$
  end for;
  unmark all nodes in $V \setminus S$ *and* S
end while;

Algorithm 4.3.2: The multiple minimum degree algorithm.

Initially, the degree of all nodes is computed. Multiple mass elimination is performed iteratively. After a node $y \in T$ is chosen arbitrarily, the algorithm determines the set Y which contains all elements in T indistinguishable from y. Once y is computed, all elements in Y and the adjacency set of Y, $adj(Y)$, are marked. Finally, set S is added to set Y and multiple mass elimination starts from the beginning with another element $y \in T$ until there are no unmarked node left. Then, the graph representation of the remaining nodes in $V \setminus S$ is computed. Simultaneously, the degrees of all marked nodes in $V \setminus S$ are updated.

For example, in Figure 4.6, the initial minimum degree δ_0 for the graph representation of the matrix is 3. The multiple minimum degree algorithm begins with $S = \{\}$, $T = \{10, 9, 8, 6, 3, 2, 1\}$. Suppose $y = 10$ is selected from T. Node 6 is indistinguishable from node 10. For $Y = \{10, 6\}$, the adjacency set includes nodes 2 and 8 which are also marked (by a '*'). After the first iteration of the algorithm, $S = \{10, 6\}$, $T = \{10^*, 9, 8^*, 6^*, 3, 2^*, 1\}$. In the second iteration, we select $y = 9$ and have $Y = \{9, 3\}$. Nodes 5 and 7 are marked because they are in the adjacency set of Y. After the end of the second iteration, we find $S = \{10, 6, 9, 3\}$ and $T = \{10^*, 9^*, 8^*, 6^*, 3^*, 2^*, 1\}$. Now only node 1 is left unmarked in T, so $y = 1$ is eliminated alone and its adjacent node 4 becomes marked. Since all the elements in T have been marked, the degree update step needs to be conducted. Figure 4.11 shows the graph representation after the elimination of nodes in S. After the degree update of the remaining nodes, the new minimum degree δ_1 is 2, $S = \{10, 6, 9, 3, 1\}$, $T = \{7, 8, 5, 2\}$. Since nodes 7 and 5, and nodes 8 and 2 are indistinguishable, they are eliminated, leaving node 4 as the last node to be processed. The reordering sequence derived from the multiple minimum degree algorithm is $P_3 = (10, 6, 9, 3, 1, 7, 5, 8, 2, 4)$. Figure 4.12 shows the filled matrix $F_{P_3 A P_3^T}$.

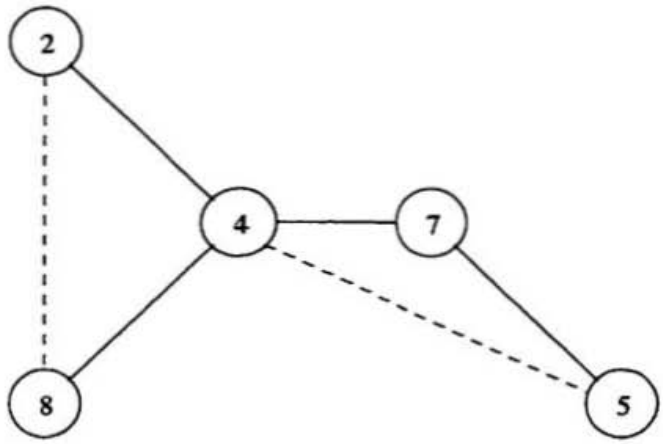


Figure 4.11: Elimination graph after first loop of multiple mass elimination.

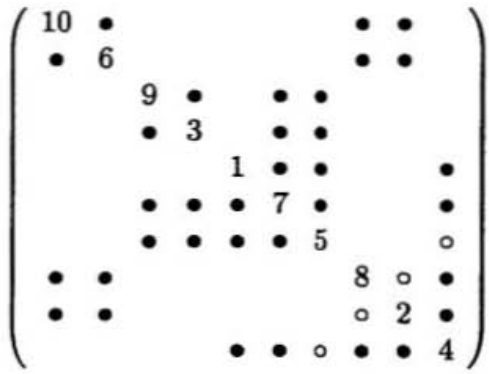


Figure 4.12: Pattern of the filled matrix F of $P_3AP_3^T$.

4.3.4 Finding the separator

After reordering the coefficient matrix into the arrowhead structure, we consider the partition of the reordered matrix into submatrices. In regard to the graph theory, the matrix partition equals partitioning of a corresponding graph to subgraphs. This procedure can be performed either by removing edges, or by deleting nodes and incident edges from the graph. As mentioned in the previous section, a small separator is desired in partitioning a matrix. Generally speaking, a separator is a small subset of nodes whose removal divides the graph approximately in half. The problem of finding the smallest separator has the same complexity as the problem of finding the minimum fill-in reordering of a sparse matrix. Both of these problems are NP hard [31].

In [20] Liu surveyed existing algorithms for finding separators. For general undirected graphs, practical heuristic partitioning methods are usually variants of the scheme developed by Kernighan and Lin [15]. The Kernighan-Lin scheme was originally designed to determine a set of small edge separators. Fiduccia and Mattheyses [10] improved the Kernighan-Lin algorithm for partitioning networks. Leiserson and Lewis [16] used the edge separators from Fiduccia's method to find node separators.

We use U and W to denote the partitions and S to denote the node separator. n is the number of nodes in the graph. If the component U satisfies $|U| + |adj(U)| < n$, the adjacent set of U forms a separator. Consider the graph shown in Figure 4.6, and let $U = \{3, 5\}$. Then $adj(U) = \{1, 7, 9\}$ and $W = \{2, 4, 6, 8, 10\}$. Removing $S = adj(U)$ from the graph results in two connected subgraphs as shown in Figure 4.13.

For $U = \{3, 5, 7, 9\}$, $adj(U) = \{1, 4\}$ and $W = \{2, 6, 8, 10\}$. If $S = adj(U)$ is removed, the graph splits into two equalized components as shown in Figure 4.14.

The example shows that an arbitrary U can lead to $adj(U)$ which may contain

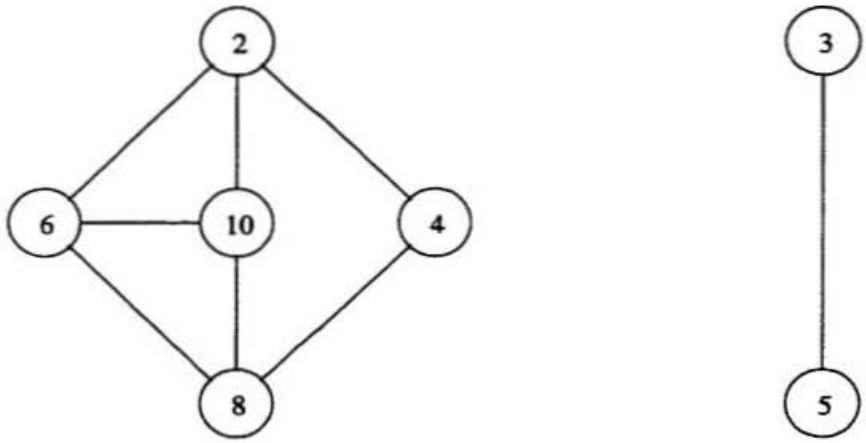


Figure 4.13: Result of removing node separator $S = \{1, 7, 9\}$.

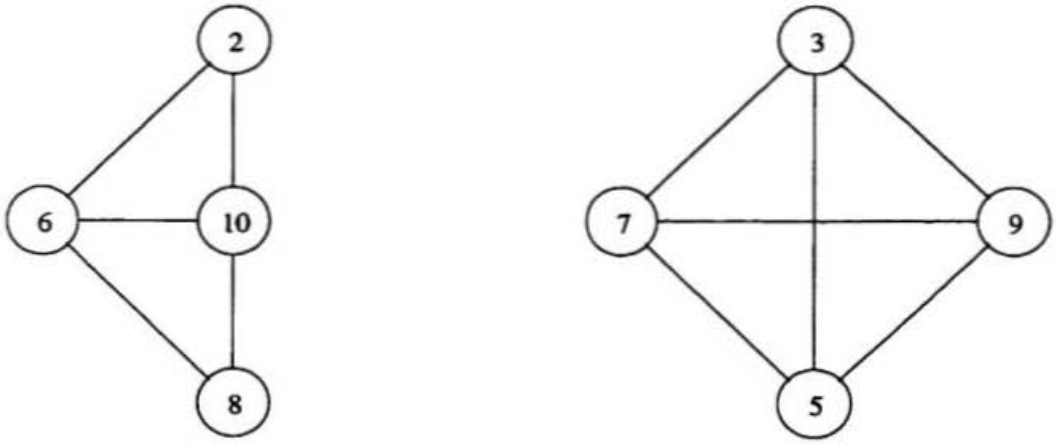


Figure 4.14: Result of removing node separator $S = \{1, 4\}$ in graph Figure 4.6.

the remaining nodes of the graph. If $|U| < |adj(U)|$, the two partitions are possibly imbalanced. **Elimination tree** can be used to find a proper separator set.

Let us consider a sparse structurally symmetric matrix A with its associate graph $G(A)$ and filled graph $G(F)$. The elimination tree $T(A)$ of a graph $G(A)$ has the same node set as $G(A)$. The commonly used data structure for representing the elimination tree is a linear array called the PARENT vector [17]. For a node i,

$$PARENT(i) = \min(j \mid j > i, l_{ij} \neq 0).$$

PARENT vector implies the father of a given node. The PARENT vector has the length of n, where $n = |V|$. Index n represents the root node which has no parent node in the elimination tree. PARENT(n) is set to 0. By using the definition of PARENT vector, we can construct the elimination tree $T(A)$ from $G(F)$ as shown in Figure 4.15. Liu [21] proved that the elimination tree $T(A)$ of $G(A)$ is a depth-first search tree of the filled graph $G(F)$ of A. The following steps implement the procedure that generates the elimination tree from $G(F)$:

Step1: According to the depth-first search, nodes with the highest index are added to the tree first. For the example, node 6 is the root of the elimination tree and the start point of the search process.

Step2: The adjacent set of node 6 contains nodes 1, 3, 4, 5. Node 5 has the highest index and therefore is selected as the descendent of node 6.

Step3: The adjacent set of node 5 is {1, 4, 6}. Node 6 has already been selected, so node 4 is the descendent of node 5.

Step4: Since node 3 is the only not-yet-visited node adjacent to node 4, node 3 is selected next.

Step5: Same as step 4, node 2 is added.

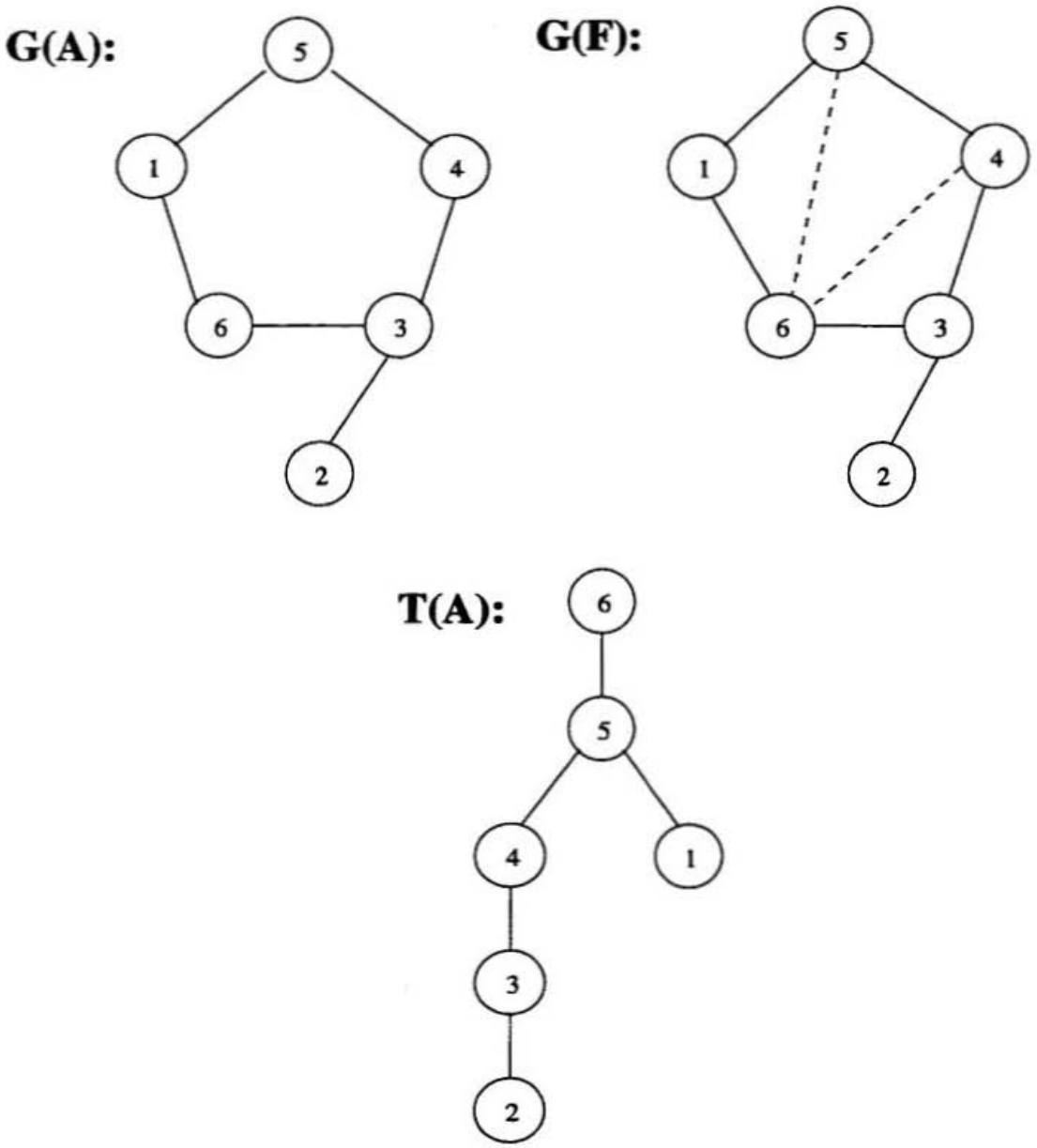


Figure 4.15: Graphs $G(A)$ and $G(F)$.

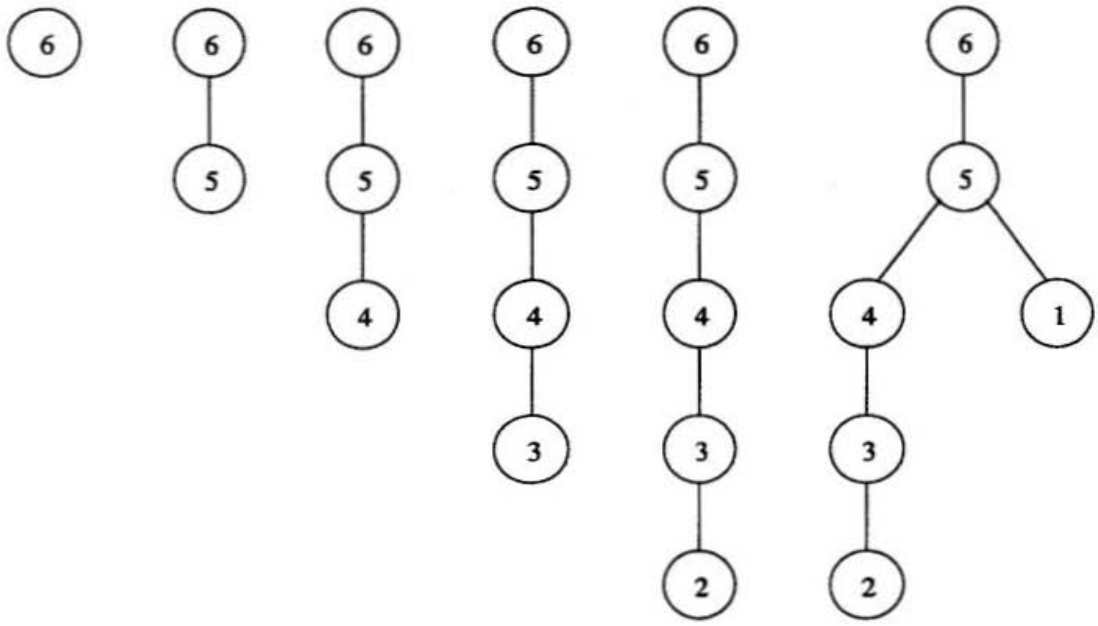


Figure 4.16: Generating the elimination tree using depth-first search.

Step6: Nodes 2, 3, 4 have no other adjacent nodes which have not been added to the tree. The depth-first search climbs back up the tree to the first node adjacent to a node which is not in the tree. Node 5 is selected, and node 1 is added as a descendent of node 5. The algorithm stops here because all nodes of the graph have been processed.

We can easily observe that the depth-first search tree shown in Figure 4.16 is identical to $T(A)$ in Figure 4.15. Algorithm 4.3.3 depicts how to use a bottom-up approach to build the elimination tree. For each node i the algorithm searches the subtree of each descendant of i (the inner "**for**" loop) until it reaches the top of the subtree (the "**while**" loop). If the parent of this top node is 0, the parent value is assigned to i. Since Algorithm 4.3.3 needs to search subtrees repeatedly to find the root node, the efficiency is rather low. If the current root of each visited node is stored in another vector, the root of the subtree containing k can be determined directly. This technique is referred to as **pass compression**. Algorithm 4.3.4 describes the improved version of

```
for i := 1 to n do
   PARENT(i) := 0;
   for each k such that k ∈ adj(i) ∧ k < i do
      r := k;
      while PARENT(r) ≠ 0 ∧ PARENT(r) ≠ i do
         r := PARENT(r)
      end while;
      if PARENT(r) = 0 then
         PARENT(r) := i
      end if
   end for
end for;
```

Algorithm 4.3.3: Computing the elimination tree.

the computing elimination tree by path compression.

```
for i := 1 to n do
   PARENT(i) := 0;
   ANCESTOR(i) := 0;
   for each k such that k ∈ adj(i) ∧ k < i do
      r := k;
      while ANCESTOR(r) ≠ 0 ∧ ANCESTOR(r) ≠ i do
         t := ANCESTOR(r);
         ANCESTOR(r) := i;
         r := t
      end while;
      if ANCESTOR(r) = 0 then
         ANCESTOR(r) := i;
         PARENT(r) := i
      end if
   end for
end for;
```

Algorithm 4.3.4: Computing elimination tree by path compression.

After the elimination tree is built, we find an initial separator by scanning the tree from the leaves to the root, looking for the node $i \in V$ which satisfies the condition

$$|T(i)| = \max\,(|T(j)| \mid |T(j)| < n/2,\, j \in V).$$

When a node i is found, we set $U = T(i)$ and $S = adj(U)$. We define $ANCESTOR(i)$ to contain all nodes from the root to i, excluding i. [19] provides an upper bound for the size of separator S:

$$|adj(T(i))| \leq |ANCESTOR(i)|.$$

This means that the size of the initial separator is restricted by the number of ancestor nodes of node i in the elimination tree. Therefore the elimination tree with minimal height is needed. Since the multiple minimum degree reordering reduces the height of the elimination tree of the original matrix [19], the separator is computed after the matrix is reordered.

For example, from the graph of the reordered matrix (with fill-ins) in Figure 4.12, we can build the elimination tree shown in Figure 4.17. $i = 9$ is selected as the root of the subtree because $|T(9)| = 4 < 5$. Then $U = \{1, 3, 5, 9\}$. The separator $S = adj(T(9)) = \{4, 7\}$ and $W = \{2, 6, 8, 10\}$. The filled matrix is reordered in order to split the original matrix into two parts, as shown in Figure 4.18.

In the obtained arrowhead structure, the block with diagonal $1, 3, 5, 9$ can be selected (see section 1) as A_1 and the block with diagonal $2, 6, 8, 10$ as A_2. If a higher level of parallelism is needed, the partitioning algorithm can be called recursively until the required number of partitions is created. Obviously, the number of partitions can only be a power of two. Therefore, the solution of linear systems requires the number of processors to be a power of two.

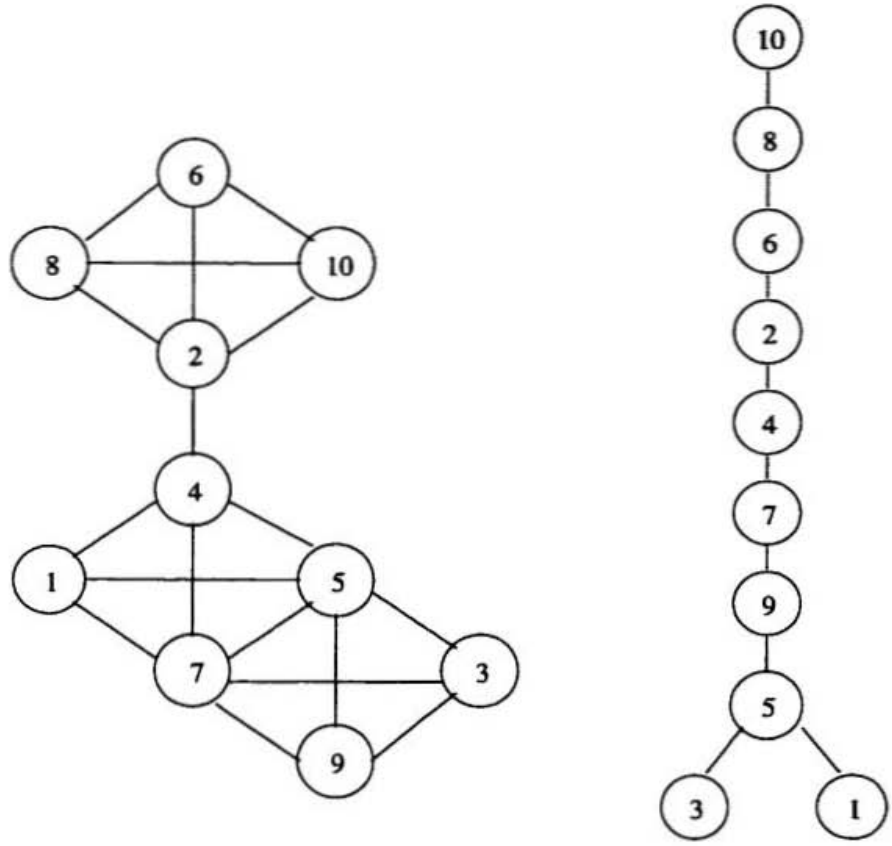


Figure 4.17: Graph representation of the reordered matrix in Figure 4.12 and the generated elimination tree.

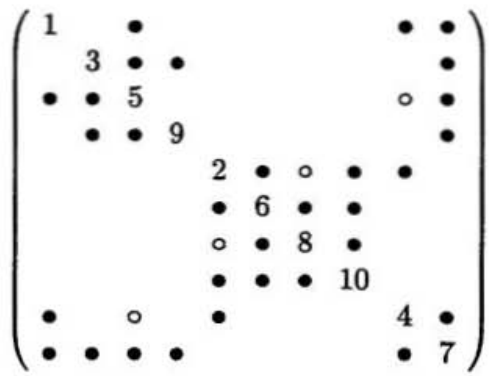


Figure 4.18: Partition of the matrix.

4.4 Summary

A domain decomposition approach to a large sparse linear system solution was described in the previous sections. The goal of this approach is to partition the matrix into a number of components (to divide the problem domain into subdomains). The slave processes receive the parts of the coefficient matrix and the right-hand vector, and then solve the local subsystems. From the literature, we can see that a domain decomposition method can make each processor work more independently. Gropp [14] discussed some issues in designing and implementing a parallel domain decomposition algorithm in general. He suggested that domain decomposition techniques will be effective on networks of workstations.

In this chapter, we also introduced the concept of graph theory and elimination trees, which are very commonly used tools for solving sparse linear systems. Variants of minimum degree algorithms were also reviewed as effective strategies for sparse matrix reordering. In the next chapter, we will describe the implementation details and result of the whole approach.

Chapter 5

Implementation, Results and Discussion

5.1 Data structures representing sparse matrices

The data structure of storing non-zero entries in large sparse matrices is very important in order to save the storage space. One standard storage scheme which is used in SPICE-PAC has three vectors to describe a coefficient matrix of a linear system: COLPTR, ROWIND and VALUES. Array VALUES stores numerical values of non-zeros of the matrix in column-wise fashion. ROWIND contains a row index of each entry in the matrix. COLPTR is an array which stores the (ROWIND and VALUES) indices of the first (nonzero) elements in consecutive columns of the original matrix.

Consider the following matrix:

$$\begin{pmatrix} 2 & 0 & 5 & -1 & 0 \\ 0 & 1 & -3 & 1 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 4 & 0 & -4 & 0 & 4 \\ 5 & -5 & 0 & 0 & 6 \end{pmatrix}$$

If columns and rows are numbered from 1 to 5, starting at the left upper corner, the matrix is represented by the data shown in Table 5.1.

Let n be the number of rows in the matrix. If the matrix is structurally symmetric,

i	1	2	3	4	5	6	7	8	9	10	11	12
COLPTR	1	4	6	9	12	13						
ROWIND	1	4	5	2	5	1	2	4	1	2	3	4
VALUES	2	4	5	1	-5	5	-3	-4	-1	1	6	4

Table 5.1: Sparse matrix format.

let n_e be the number of off-diagonal non-zero entries in the upper (lower) half. The amount of storage required to represent the coefficient matrix in this standard storage format is:

$$|COLPTR| + |ROWIND| + |VALUES| = (n+1) + 2*(n+2*n_e)$$
$$= 3*n + 4*n_e + 1.$$

5.2 Performance

In this section, we discuss the performance of the master-slave approach to the solution of large sparse linear systems. The linear systems used as examples are from the Harwell-Boeing Sparse Matrix Collection and they are characterized in Table 5.2. The

Name	n	n_e	density
MATRIX1	216	4374	9.38%
MATRIX2	839	22613	3.21%

Table 5.2: Description of tested linear systems.

environment is a cluster of SUN workstations connected with Ethernet and the network topology is a multistage network where one workstation in the network cannot communicate with any other workstation directly, but must do so through repeaters or/and switches. All the leaf workstations communicate through switches and switches

are connected by repeaters. Figure 5.1 shows the speedup gained for MATRIX1 and MATRIX2.

Figure 5.1 shows that we did not gain speedup through the domain decomposition approach when using two workstations. However, if we used four or eight workstations, we can gain a moderate speedup.

For the analysis, we assume a workstation cluster with N nodes. One of the N nodes is the master node which performs the matrix partitioning. If T_i denotes the execution time for node i, we have:

$$T_i = T_i^{comm} + T_i^{comp} + T_i^{A_s} + T_{serial}$$

where:

T_i^{comm} = communication time between master node and node i, or other slave nodes and node i,

T_i^{comp} = computation time for solving $\tilde{M}_i$, $\tilde{N}_i$ and x_i on node i,

$T_i^{A_s}$ = computation time of x_s on node i,

T_{serial} = serial execution time (matrix partition and computation of A_s) on master node.

The total time for parallel execution is the same as the maximum total node time, which is:

$$T_p = \max(T_i^{comm} + T_i^{comp} + T_i^{A_s}) + T_{serial}\ 2 \leq i \leq N.$$

The serial computation time is:

$$T_s = T_{serial} + T_{A_1}$$

where T_A is the time of solving the linear system as one block on a single machine. We assume the workload is the same on each node, so that the speedup can be expressed

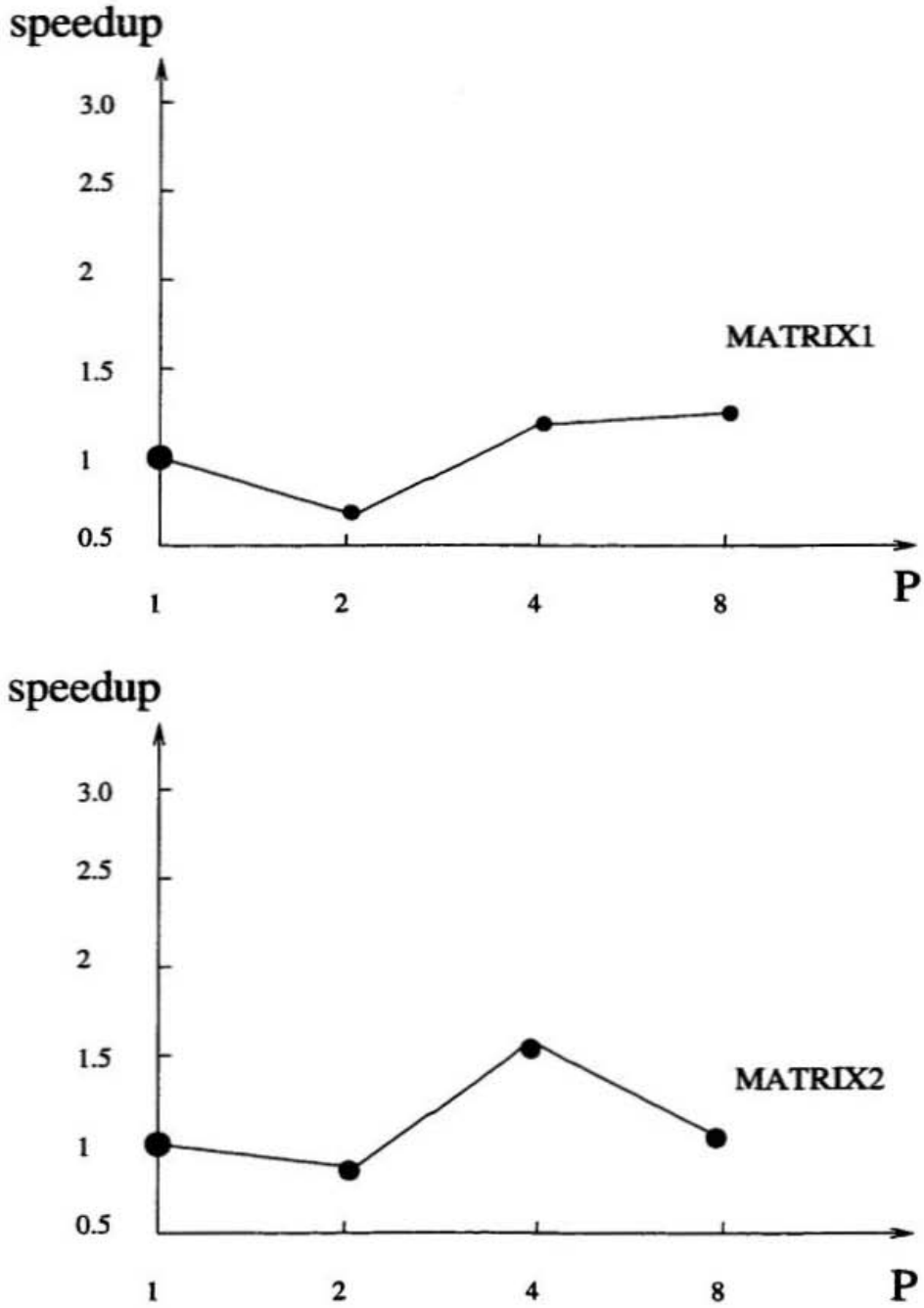


Figure 5.1: Performance of test problems on network of workstations.

as:

$$speedup = \frac{T_s}{T_p} = \frac{T_{serial} + T_A}{T_i^{comm} + T_i^{comp} + T_i^{A_s} + T_{serial}}. \quad (5.1)$$

When the number of workstations is increased from one to two, four and eight, the coefficient matrix is split into 2, 4 and 8 blocks with separator A_s respectively. Roughly, the partition reduces the size of the linear system by half as the number of processors increases. T_i^{comp} includes the time of computing $x_i + \tilde{M}_i x_s = \tilde{b}_i$, $A_i \tilde{M}_i = M_i$ and $A_i \tilde{b}_i = b_i$. A_i decreases at nearly a power of two, if A_s is very small.

Data collected after matrix partitioning shows that the size of the separator A_s does not grow linearly (see Table 5.3). Therefore, the speedup observed is influenced by load imbalance and communication overhead, as shown in Figure 5.2. [ht]

	# proc.	$\|A_s\|$	*partitions*	*time*
MATRIX1	2	21	92 103	0.35s
	4	46	39 48 27 56	0.62s
	8	63	22 14 25 20 17 19 16 20	0.76s
MATRIX2	2	48	387 404	0.80s
	4	75	162 201 220 181	1.28s
	8	98	95 65 101 78 62 132 86 122	1.66s

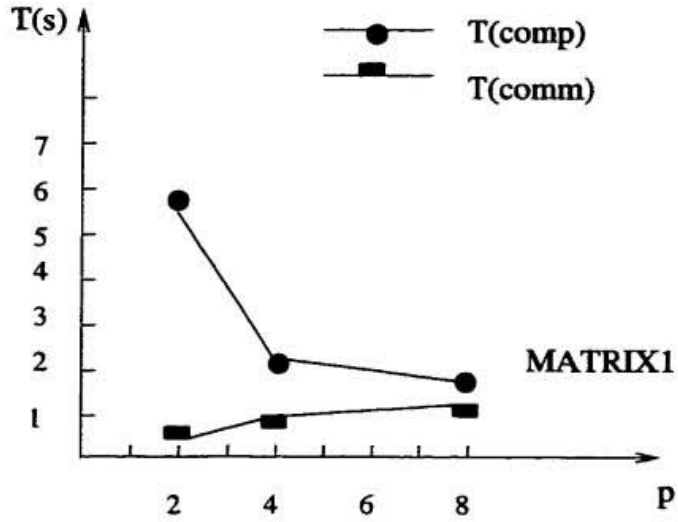
Table 5.3: Partitioning of MATRIX1 and MATRIX2.

There are three kinds of data communication conducted in this master-slave model. One communication is to broadcast the matrix partitions to slave nodes. Another one is to exchange the structure $N_i \tilde{M}_i$ in order to calculate the Schur complement of A_s to every other node. The third one is to return x_i to the master node. When 8 processors are used, communication becomes more important. The overall performance on eight processors is more influenced by communication time and network speed than in the

case of 4 or 2 nodes.

In our experiments with two processors, the total execution time is even greater than for a single processor. The reason is that the computations $A_i\tilde{M}_i = M_i$ and $A_i\tilde{b}_i = b_i$ introduce extra workload in solving linear systems. As more processors are used, the influence of the extra work is overwhelmed by the decreasing size of blocks assigned to each processor. Therefore a slight speedup was gained for 4 and 8 processors.

The unexpected drop of speedup for MATRIX2 on eight processors is caused by imbalance of the matrix partitioning. The performance of eight processors is more sensitive to the size of each partition.



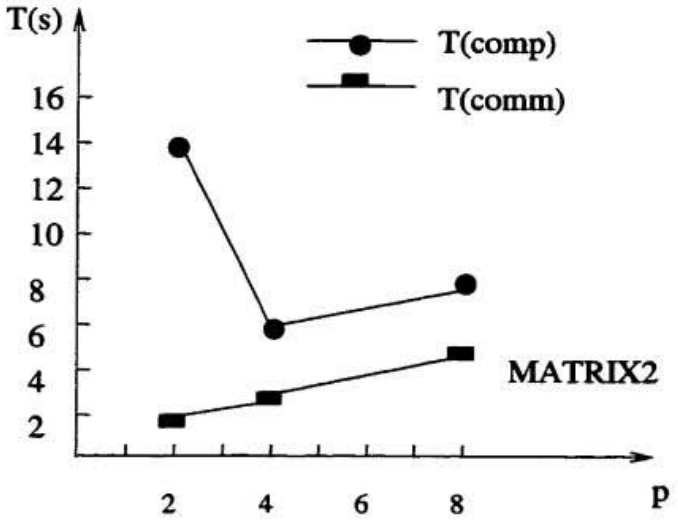


Figure 5.2: Computation and communication time.

Chapter 6

Conclusions

This thesis is devoted to efficient concurrent solutions of large sparse linear systems. The motivation of the project and a review of the related research are given. The main goal of this work is to find the most suitable approach to solving large sparse linear systems on networks of workstations.

Because of the high communication overhead that is typical for networks of workstations, algorithms with low frequency of communication are needed. One attractive category of these algorithms is the domain decomposition method. The idea of domain decomposition is to split the problem domain into a number of smaller subdomains and to use different processors to process the subdomains in parallel.

The domain decomposition approach applied to the solution of large sparse linear systems is studied in detail. First, the coefficient matrix is reordered into the "arrowhead" structure. Each parallel process is assigned to a subblock in the "arrowhead" matrix. Then the Schur complement is calculated, and the corresponding unknowns are determined through a back substitution process. The computation of the Schur complement and solution of the Schur complement are major tasks introduced by the domain decomposition method, and degrade the overall performance. This is clearly illustrated

by the two examples. However, as more workstations are used, communication time becomes more important. If a much faster network is available, better speedups are expected on 4 to 8 workstations. Furthermore, as the number of partitions increases, load imbalance can become a major issue. For MATRIX2, when 8 workstations were used, the whole computation was delayed by the slowest workstation. Also, the performance was influenced in this case by different computation speeds of workstations.

To enhance the performance, some improvements can be recommended. First, a better algorithm for matrix partitioning needs to be developed. The separators in the test cases are rather large and the partitioning consumes a considerable amount of time. Second, load balancing is very important for a better performance. However, in order to balance the workload of each processor, some additional time is required. It is an important issue for parallel computation on networks of workstations to find a reasonable tradeoff between the quality of balancing the load and the time overhead needed for this balancing. Third, another level of parallelization can be explored. Gilbert [13] proposed a parallel graph partitioning algorithm for message-passing architectures. When the coefficient matrix is very large, the time required to partition the matrix can be quite significant and also a single workstation may have insufficient memory for the partitioning process.

The hybrid approach needs to be explored carefully for use in circuit simulation programs, because, in this case, sequences of linear systems with similar sparsity structure need to be solved. Since the parallelization of iterative solvers of large sparse linear systems is straightforward, a combination of the direct technique using domain decomposition and an iterative technique using the solution from the domain decomposition as the starting point vector can result in good speedups in a network environment.

Bibliography

[1] G. Alaghband, "Parallel Pivoting Combined with Parallel Reduction and Fill-in Control", *Parallel Computing*, 11, pp.201-221, Nov. 1989.

[2] G. Alaghband and H. Jordan, "Sparse Gaussian Elimination with Controlled Fill-in on a Shared Memory Multiprocessor", *IEEE Trans. Comput.*, 38, pp.1539-1557, Nov. 1989.

[3] G. Alaghband, "Parallel Sparse Matrix Solution and Performance", *Parallel Computing*, 21, pp.1407-1430, 1995.

[4] P. Bjorstad, W. Coughran and E. Grosse, "Parallel domain decomposition applied to coupled transport equations", *Proc. 7th Domain Decomposition Conference*, Penn State, Oct. 1993.

[5] P. Carlin, "Distributed linear algebra on networks of workstations", Master's Thesis, California Institute of Technology, Pasadena, California, July 1995.

[6] I. Duff, A. Erisman and J. Reid, "Direct Methods for Sparse Matrices", Oxford Science Publications, 1986.

[7] I. Duff, "Sparse Numerical Linear Algebra: Direct Methods and Preconditioning", Technical Report TR/PA/96/22.

[8] J. Dongarra, I. Duff, D. Sorensen and H. van der Vorst, "Solving Linear Systems on Vector and Shared Memory Computers", SIAM Publ., 1991.

[9] U. Feldmann, K. Rauh and K. Steger, "Circuit Simulation on Vectorprocessors", *1st International Conference on Computer Technology, Systems and Applications*, May, 1987.

[10] C. Fiduccia and R. Mattheyses, "A Linear-time Heuristic for Improving Network Partition", *Proceedings of the 19th Design Automation Conference*, pp.175-181, ACM, 1982.

[11] A. George and J. Liu, "The Evolution of the Minimum-Degree Ordering Algorithm", *SIAM Review*, 31(1), pp.1-19, 1989.

[12] A. George and D. McIntyre, "On the Application of the Minimum Degree Algorithm to Finite Element Systems", *SIAM Journal on Numerical Analysis*, 15(1), pp.90-112, 1978.

[13] J. Gilbert and E. Zmijewski, "A Parallel Graph Partitioning Algorithm for a Message-passing Multiprocessor", *International Journal of Parallel Programming*, 16(6), pp.1-19, 1987.

[14] W. Gropp, "Parallel Computing and Domain Decomposition", *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pp.349-361, SIAM Publ., 1991.

[15] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Syst. Tech. J.*, 49, pp.291-307, 1970.

[16] C. Leiserson and J. Lewis, "Orderings for Parallel Sparse Symmetric Factorization", *Third SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, 1987.

[17] A. Liegmann, "Efficient Solution of Large Sparse Linear Systems", *Ph. D. Dissertation*, Swiss Federal Institution of Technology, Zurich, 1995.

[18] J. Liu, "Modification of the Minimum-Degree Algorithm by Multiple Elimination", *ACM Transactions on Mathematical Software*, 11(2), pp.141-153, 1985.

[19] J. Liu, "Equivalent Sparse Matrix Reordering by Elimination Tree Rotations", *SIAM Journal on Scientific and Statistical Computing*, 9(3), pp.424-444, 1988.

[20] J. Liu, "A Graph Partitioning Algorithm by Node Separators", *ACM Transactions on Mathematical Software*, 15(3), pp.198-219, 1989.

[21] J. Liu, "The Role of Elimination Trees in Sparse Factorization", *SIAM Journal on Matrix Analysis and Applications*, 11(1), pp.134-172, 1990.

[22] G. Lo and Y. Saad, "Iterative Solution of General Sparse Linear Systems on Clusters of Workstations", *http://www.cs.umn.edu/research/darpa/p_sparslib/psp-abs.html.*

[23] Y. Saad, "A Flexible Inner-outer Preconditioned GMRES Algorithm", *SIAM Journal on Scitific and Statistical Computing*, 14, pp.461-469, 1993.

[24] Y. Saad and A. Malevsky, "PSPARSLIB: A Portable Library of Distributed Memory Sparse Iterative Solvers", *http://www.cs.umn.edu/research/darpa/p_sparslib/psp-abs.html*, May 1995.

[25] Y. Saad, "Iterative Solution of General Sparse Linear Systems on Clusters of Workstations", *http://www.cs.umn.edu/research/darpa/p_sparslib/psp-abs.html*, August 1996.

[26] P. Sadayappan and S.K. Rao , "Communication Reduction for Distributed Sparse Matrix Factorization on a Processor Mesh", *Proc. Supercomputing'89*, pp.371-379, ACM Press, 1989.

[27] A. Skjellum, "Concurrent dynamic simulation: Multicomputer algorithms research applied to ordinary differential-algebraic process systems in chemical engineering", Ph. D. thesis, California Institute of Technology, Pasadena, CA, May 1990.

[28] A. Stappen, Rob Bisseling and J.G.G Vorst, "Parallel sparse LU decomposition on a mesh network of Transputers", *SIAM J. Matrix Anal. Appl*, 14, 3, pp.853-879, July 1993.

[29] J. Swart and J. Blom, "Experiences with Sparse Matrix Solvers in Parallel ODE Software", *Computers Math. Appl.*, 31(9), pp.43-45,1996.

[30] U.Wever and Q. Zheng, "Domain Decomposition Methods for Circuit Simulation", *Proc. of the 8-th Workshop on Parallel and Distributed Simulation (PADS'94)*, San Diego, CA, July 6-8, 1994.

[31] M. Yannakakis, "Computing the Minimum Fill-in Is NP-complete", *SIAM Journal on Algebraic Discrete Methods*, 2(1), pp.77-79, 1981.

[32] Z. Zlatev, " PARASPAR: A package for the solution of large and sparse systems of linear algebraic equations on parallel computers with shared memory", *http://www.wi.leidenuniv.nl/CS/HPC/apparc-deliverables/*.

[33] Z. Zlatev, J. Wasniewski and K. Schaumburg, "Y12M - Solution of Large and Sparse Systems of Linear Algebraic Equations", *Lecture Notes in Computer Science*, 121, Springer Verlag, 1981.

[34] W. Zuberek, " SPICE-PAC version 2G6c: An Overview", Technical Report 8903, Department of Computer Science, Memorial University of Newfoundland, March 1989.

[35] LAPACK Working Note 95, "ScaLAPACK: A portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance", *http://www.ciril.fr/CIRIL/Logiciels/scalapack.docs/*.

Appendix A

The Results of the SPICE-PAC Profile

A.1 The results from "test-adder"

The results of the profile for "test-adder" circuit are as follows:

```
granularity: each sample hit covers 4 byte(s) for 0.01% of 8.20 seconds

index  %time    self descendents  called/total       parents
                                  called+self    name     index
                                  called/total       children

                0.00        8.20       1/1           main [2]
[1]     100.0   0.00        8.20       1         sppac_ [1]
                0.00        7.95       1/1           spicer_ [3]
                0.00        0.14       1/1           spicec_ [20]
                0.00        0.12       1/1           spicea_ [25]
                0.00        0.00       1/1           outres_ [86]
                0.00        0.00       1/1           extime_ [89]
                0.00        0.00       1/1           spicej_ [92]
                0.00        0.00       1/1           otpfil_ [107]
                0.00        0.00       1/1           spicem_ [108]
-----------------------------------------------
                                                 <spontaneous>
[2]     100.0   0.00        8.20                  main [2]
                0.00        8.20       1/1           sppac_ [1]
-----------------------------------------------
                0.00        7.95       1/1           sppac_ [1]
[3]      96.9   0.00        7.95       1         spicer_ [3]
                0.00        7.89       1/1           spptra_ [4]
                0.00        0.05       2/2           sppdca_ [32]
                0.00        0.00       1/1           sppres_ [59]
                0.00        0.00       1/1           sppbpt_ [78]
                0.00        0.00       1/29          spmg08_ [54]
                0.00        0.00       1/6829        spmptr_ [19]
                0.00        0.00       2/22          sppxsg_ [99]
                0.00        0.00       2/22          sppxtp_ [100]
                0.00        0.00       1/1           sppchs_ [113]
```

```
                0.00        0.00        1/1          sppior_ [115]
-------------------------------------------------------------------
                0.00        7.89        1/1          spicer_ [3]
[4]     96.3    0.00        7.89        1          spptra_ [4]
                0.00        7.82      252/252        spptrx_ [5]
                0.00        0.03     1265/1265       spmrmp_ [36]
                0.00        0.03       15/342        spptct_ [11]
                0.00        0.00       66/4057       spmext_ [18]
                0.00        0.00      256/1671       spxc08_ [49]
                0.00        0.00      253/253        sppstr_ [69]
                0.00        0.00        5/29         spmg08_ [54]
                0.00        0.00       10/6829       spmptr_ [19]
                0.00        0.00        7/50         spmclr_ [66]
                0.00        0.00        1/5          spmcrn_ [76]
                0.00        0.00       21/305088     spptst_ [17]
                0.00        0.00        2/5902       spptim_ [53]
-------------------------------------------------------------------
                0.00        7.82      252/252        spptra_ [4]
[5]     95.4    0.00        7.82      252          spptrx_ [5]
                0.10        7.03      332/334        sppitr_ [6]
                0.02        0.65      327/342        spptct_ [11]
                0.01        0.00      332/334        sppsor_ [52]
                0.00        0.00      332/336        sppcje_ [60]
                0.00        0.00      332/332        sppccf_ [62]
                0.00        0.00      659/305088     spptst_ [17]
                0.00        0.00        2/1671       spxc08_ [49]
-------------------------------------------------------------------
                0.00        0.04        2/334        sppdca_ [32]
                0.10        7.03      332/334        spptrx_ [5]
[6]     87.5    0.10        7.08      334          sppitr_ [6]
                0.06        4.62     1551/1551       sppldm_ [7]
                1.36        0.00     1217/1217       sppdcs_ [9]
                0.07        0.97     1217/1217       sppdcd_ [10]
                0.01        0.00     1217/1671       spxc08_ [49]
                0.00        0.00        2/1186       spmsiz_ [39]
-------------------------------------------------------------------
                0.06        4.62     1551/1551       sppitr_ [6]
[7]     57.0    0.06        4.62     1551         sppldm_ [7]
                3.25        0.55     1551/1552       sppbjt_ [8]
                0.55        0.05     1551/1552       sppjdd_ [13]
                0.14        0.00    63591/63591      sppldu_ [21]
                0.00        0.06     1551/1726       spxz08_ [29]
                0.01        0.00     1551/1551       sppdsr_ [50]
                0.00        0.00     3102/5902       spptim_ [53]
                0.00        0.00      140/1186       spmsiz_ [39]
                0.00        0.00     1551/1552       sppmos_ [58]
                0.00        0.00     1551/305088     spptst_ [17]
                0.00        0.00     1551/1552       sppfet_ [95]
-------------------------------------------------------------------
                0.00        0.00        1/1552       sppdca_ [32]
                3.25        0.55     1551/1552       sppldm_ [7]
[8]     46.4    3.25        0.55     1552         sppbjt_ [8]
                0.36        0.14   224038/241372     sppitg_ [15]
                0.05        0.00    94766/113063     sppilm_ [30]
-------------------------------------------------------------------
                1.36        0.00     1217/1217       sppitr_ [6]
[9]     16.6    1.36        0.00     1217         sppdcs_ [9]
                0.00        0.00     1217/162356     sppixx_ [16]
```

```
                0.07        0.97     1217/1217          sppitr_ [6]
[10]     12.6   0.07        0.97     1217           sppdcd_ [10]
                0.56        0.04   137521/137521        spp_dcd_ [14]
                0.34        0.00   155520/162356        sppixx_ [16]
                0.02        0.00      100/103           sppswp_ [43]
                0.00        0.01        3/4             sppmtl_ [46]
                0.00        0.00     2434/5902          spptim_ [53]
-----------------------------------------------
                0.00        0.03       15/342           spptra_ [4]
                0.02        0.65      327/342           spptrx_ [5]
[11]     8.5    0.02        0.68      342           spptct_ [11]
                0.64        0.03    55404/55404         sppter_ [12]
                0.00        0.00    342/305088          spptst_ [17]
-----------------------------------------------
                0.64        0.03    55404/55404         spptct_ [11]
[12]     8.3    0.64        0.03    55404           sppter_ [12]
                0.02        0.01    55404/305088        spptst_ [17]
-----------------------------------------------
                0.00        0.00       1/1552           sppdca_ [32]
                0.55        0.05     1551/1552          sppldm_ [7]
[13]     7.3    0.55        0.05     1552           sppjdd_ [13]
                0.03        0.01    17334/241372        sppitg_ [15]
                0.01        0.00    18297/113063        sppjlm_ [30]
-----------------------------------------------
                0.56        0.04   137521/137521        sppdcd_ [10]
[14]     7.3    0.56        0.04   137521          spp_dcd_ [14]
                0.00        0.03     1914/1914          spprsx_ [37]
                0.00        0.01      118/4057          spmext_ [18]
```

A.2 The results from "test-adder-2"

The results of the profile for "test-adder-2" circuit are as follows:

```
granularity: each sample hit covers 4 byte(s) for 0.00% of 38.20 seconds

                                called/total       parents
index  %time    self descendents  called+self    name     index
                                  called/total       children

                0.00       38.20       1/1           main [2]
[1]      100.0  0.00       38.20       1         sppac_ [1]
                0.00       37.93       1/1           spicer_ [3]
                0.00        0.14       1/1           spicec_ [26]
                0.00        0.12       1/1           spicea_ [30]
                0.00        0.00       1/1           outres_ [63]
                0.00        0.00       1/1           extime_ [91]
                0.00        0.00       1/1           spicej_ [93]
                0.00        0.00       1/1           spicem_ [94]
                0.00        0.00       1/1           otpfil_ [109]
-----------------------------------------------
                                                     <spontaneous>
[2]      100.0  0.00       38.20                 main [2]
                0.00       38.20       1/1           sppac_ [1]
-----------------------------------------------
```

```
                0.00       37.93        1/1          sppac_ [1]
[3]     99.3    0.00       37.93        1          spicer_ [3]
                0.02       37.86        1/1          spptra_ [4]
                0.00        0.05        2/2          sppdca_ [34]
                0.00        0.00        1/1          sppbpt_ [64]
                0.00        0.00        1/1          sppres_ [67]
                0.00        0.00        1/29         spmg08_ [61]
                0.00        0.00        2/22         sppxsg_ [77]
                0.00        0.00        2/22         sppxtp_ [78]
                0.00        0.00        1/11880      spmptr_ [21]
                0.00        0.00        1/1          sppchs_ [114]
                0.00        0.00        1/1          sppior_ [116]
-----------------------------------------------
                0.02       37.86        1/1          spicer_ [3]
[4]     99.2    0.02       37.86        1          spptra_ [4]
                0.02       37.45     1212/1212       spptrx_ [5]
                0.01        0.19     6065/6065       spmrmp_ [25]
                0.00        0.15       68/1650       spptct_ [11]
                0.00        0.01      302/4308       spmext_ [24]
                0.01        0.00     1213/1213       sppstr_ [52]
                0.01        0.00     1216/7056       spxc08_ [33]
                0.00        0.00        5/29         spmg08_ [61]
                0.00        0.00       10/11880      spmptr_ [21]
                0.00        0.00        7/50         spmclr_ [71]
                0.00        0.00        1/5          spmcrn_ [87]
                0.00        0.00      101/1456767    spptst_ [17]
                0.00        0.00        2/27406      spptim_ [38]
-----------------------------------------------
                0.02       37.45     1212/1212       spptra_ [4]
[5]     98.1    0.02       37.45     1212         spptrx_ [5]
                0.55       33.32     1600/1602       sppitr_ [6]
                0.10        3.43     1582/1650       spptct_ [11]
                0.03        0.00     1600/1602       sppsor_ [41]
                0.01        0.00     1600/1604       sppcje_ [50]
                0.00        0.01     1600/1600       sppccf_ [57]
                0.00        0.00     3182/1456767    spptst_ [17]
                0.00        0.00        2/7056       spxc08_ [33]
                0.00        0.00        1/1          spplim_ [117]
-----------------------------------------------
                0.00        0.04        2/1602       sppdca_ [34]
                0.55       33.32     1600/1602       spptrx_ [5]
[6]     88.8    0.55       33.36     1602         sppitr_ [6]
                0.33       22.04     7244/7244       sppldm_ [7]
                6.48        0.01     5642/5642       sppdcs_ [9]
                0.32        4.12     5642/5642       sppdcd_ [10]
                0.06        0.00     5642/7056       spxc08_ [33]
                0.00        0.00        2/1186       spmsiz_ [40]
-----------------------------------------------
                0.33       22.04     7244/7244       sppitr_ [6]
[7]     58.6    0.33       22.04     7244         sppldm_ [7]
               15.59        2.72     7244/7245       sppbjt_ [8]
                2.43        0.23     7244/7245       sppjdd_ [15]
                0.65        0.00   297004/297004     sppldu_ [18]
                0.00        0.34     7244/7508       spxz08_ [20]
                0.03        0.00     7244/7244       sppdsr_ [43]
                0.02        0.00    14488/27406      spptim_ [38]
                0.00        0.00     7244/7245       sppfet_ [59]
```

```
                0.00        0.00     7244/7245          sppmos_ [60]
                0.00        0.00     7244/1456767       spptst_ [17]
                0.00        0.00      140/1186          spmsiz_ [40]
-----------------------------------------------------------------
                0.00        0.00        1/7245          sppdca_ [34]
               15.59        2.72     7244/7245          sppldm_ [7]
[8]     47.9   15.60        2.72     7245            sppbjt_ [8]
                1.82        0.69  1086000/1170217       sppitg_ [13]
                0.21        0.00   444358/529538        sppjlm_ [22]
-----------------------------------------------------------------
                6.48        0.01     5642/5642          sppitr_ [6]
[9]     17.0    6.48        0.01     5642            sppdcs_ [9]
                0.01        0.00     5642/666806        sppixx_ [16]
-----------------------------------------------------------------
                0.32        4.12     5642/5642          sppitr_ [6]
[10]    11.6    0.32        4.12     5642            sppdcd_ [10]
                2.64        0.04   637546/637546        spp_dcd_ [14]
                1.40        0.00   655545/666806        sppixx_ [16]
                0.02        0.00    11284/27406         spptim_ [38]
                0.00        0.01        3/4             sppmtl_ [49]
                0.01        0.00      100/103           sppswp_ [55]
-----------------------------------------------------------------
                0.00        0.15       68/1650          spptra_ [4]
                0.10        3.43     1582/1650          spptrx_ [5]
[11]     9.6    0.11        3.58     1650            spptct_ [11]
                3.41        0.17   267300/267300        sppter_ [12]
                0.00        0.00     1650/1456767       spptst_ [17]
-----------------------------------------------------------------
                3.41        0.17   267300/267300        spptct_ [11]
[12]     9.4    3.41        0.17   267300            sppter_ [12]
                0.13        0.04   267300/1456767       spptst_ [17]
-----------------------------------------------------------------
                0.14        0.05    84217/1170217       sppjdd_ [15]
                1.82        0.69  1086000/1170217       sppbjt_ [8]
[13]     7.1    1.96        0.75  1170217           sppitg_ [13]
                0.58        0.17  1170217/1456767       spptst_ [17]
-----------------------------------------------------------------
                2.64        0.04   637546/637546        sppdcd_ [10]
[14]     7.0    2.64        0.04   637546            spp_dcd_ [14]
                0.01        0.03     1914/1914          spprsx_ [39]
                0.00        0.01      118/4308          spmext_ [24]
```

A.3 The results from "test-add-2"

The results of the profile for "test-add-2" circuit are as follows:

```
granularity: each sample hit covers 4 byte(s) for 0.00% of 62.99 seconds

                                        called/total       parents
index  %time    self descendents  called+self    name     index
                                        called/total       children

                0.00       62.99       1/1             main [2]
[1]     100.0   0.00       62.99       1               sppac_ [1]
                0.00       62.29       1/1             spicer_ [3]
```

```
                0.00        0.40        1/1           spicec_ [23]
                0.00        0.30        1/1           spicea_ [29]
                0.00        0.00        1/1           outres_ [62]
                0.00        0.00        1/1           extime_ [91]
                0.00        0.00        1/1           spicej_ [96]
                0.00        0.00        1/1           otpfil_ [107]
                0.00        0.00        1/1           spicem_ [108]
-----------------------------------------------
                                                      <spontaneous>
[2]     100.0   0.00       62.99                  main [2]
                0.00       62.99        1/1           sppac_ [1]
-----------------------------------------------
                0.00       62.29        1/1           sppac_ [1]
[3]      98.9   0.00       62.29        1          spicer_ [3]
                0.02       62.16        1/1           spptra_ [4]
                0.00        0.10        2/2           sppdca_ [38]
                0.00        0.00        1/1           sppres_ [68]
                0.00        0.00        1/1           sppbpt_ [70]
                0.00        0.00        1/31          spmg08_ [58]
                0.00        0.00        1/15780       spmptr_ [19]
                0.00        0.00        2/26          sppxsg_ [101]
                0.00        0.00        2/26          sppxtp_ [102]
                0.00        0.00        1/1           sppchs_ [114]
                0.00        0.00        1/1           sppior_ [116]
-----------------------------------------------
                0.02       62.16        1/1           spicer_ [3]
[4]      98.7   0.02       62.16        1          spptra_ [4]
                0.03       61.70      940/940         spptrx_ [5]
                0.00        0.19     4705/4705        spmrmp_ [32]
                0.00        0.18      43/1291         spptct_ [11]
                0.03        0.00     941/941          sppstr_ [50]
                0.02        0.00     944/6112         spxc08_ [37]
                0.00        0.01     209/8277         spmext_ [20]
                0.00        0.00       5/31           spmg08_ [58]
                0.00        0.00      10/15780        spmptr_ [19]
                0.00        0.00       7/70           spmclr_ [65]
                0.00        0.00       1/5            spmcrn_ [82]
                0.00        0.00      71/2268500      spptst_ [17]
                0.00        0.00       2/23065        spptim_ [45]
-----------------------------------------------
                0.03       61.70      940/940         spptra_ [4]
[5]      98.0   0.03       61.70      940          spptrx_ [5]
                0.74       55.67     1257/1259        sppitr_ [6]
                0.14        5.09     1248/1291        spptct_ [11]
                0.02        0.00     1257/1259        sppsor_ [52]
                0.01        0.00     1257/1261        sppcje_ [54]
                0.01        0.01     1257/1257        sppccf_ [55]
                0.00        0.00     2505/2268500     spptst_ [17]
                0.00        0.00       2/6112         spxc08_ [37]
-----------------------------------------------
                0.00        0.09       2/1259         sppdca_ [38]
                0.74       55.67     1257/1259        spptrx_ [5]
[6]      89.7   0.74       55.76     1259          sppitr_ [6]
                0.48       35.53     6072/6072        sppldm_ [7]
               11.34        0.01     4813/4813        sppdcs_ [9]
                0.57        7.74     4813/4813        sppdcd_ [10]
                0.08        0.00     4813/6112        spxc08_ [37]
                0.00        0.00       2/2295         spmsiz_ [39]
```

```
-----------------------------------------------
                0.48       35.53     6072/6072         sppitr_ [6]
[7]     57.2    0.48       35.53     6072          sppldm_ [7]
               25.14        4.19     6072/6073         sppbjt_ [8]
                4.29        0.33     6072/6073         sppjdd_ [14]
                1.04        0.00   479688/479688       sppldu_ [18]
                0.01        0.48     6072/6423         spxz08_ [22]
                0.02        0.00    12144/23065        spptim_ [45]
                0.01        0.00     6072/6072         sppdsr_ [56]
                0.01        0.00     6072/6073         sppmos_ [59]
                0.00        0.01      176/2295         spmsiz_ [39]
                0.00        0.00     6072/6073         sppfet_ [60]
                0.00        0.00     6072/2268500      spptst_ [17]
-----------------------------------------------
                0.00        0.00        1/6073         sppdca_ [38]
               25.14        4.19     6072/6073         sppldm_ [7]
[8]     46.6   25.15        4.19     6073          sppbjt_ [8]
                2.79        1.09  1708074/1828464      sppitg_ [15]
                0.31        0.00   640018/762646       sppjlm_ [27]
-----------------------------------------------
               11.34        0.01     4813/4813         sppitr_ [6]
[9]     18.0   11.34        0.01     4813          sppdcs_ [9]
                0.01        0.00     4813/1170347      sppixx_ [16]
-----------------------------------------------
                0.57        7.74     4813/4813         sppitr_ [6]
[10]    13.2    0.57        7.74     4813          sppdcd_ [10]
                4.92        0.12  1082925/1082925      spp_dcd_ [13]
                2.59        0.00  1154346/1170347      sppixx_ [16]
                0.08        0.00      323/328          sppswp_ [41]
                0.00        0.02        3/4            sppmtl_ [49]
                0.02        0.00     9626/23065        spptim_ [45]
-----------------------------------------------
                0.00        0.18       43/1291         spptra_ [4]
                0.14        5.09     1248/1291         spptrx_ [5]
[11]    8.6     0.15        5.26     1291          spptct_ [11]
                4.99        0.27   418284/418284       sppter_ [12]
                0.00        0.00     1291/2268500      spptst_ [17]
-----------------------------------------------
                4.99        0.27   418284/418284       spptct_ [11]
[12]    8.4     4.99        0.27   418284          sppter_ [12]
                0.20        0.06   418284/2268500      spptst_ [17]
```

A.4 The results from "test-inv-1"

The results of the profile for "test-inv-1" circuit are as follows:

```
granularity: each sample hit covers 4 byte(s) for 0.06% of 1.64 seconds

index  %time    self  descendents  called/total       parents
                                   called+self    name     index
                                   called/total       children

                0.00        1.64        1/1           main [2]
[1]     100.0   0.00        1.64        1         sppac_ [1]
                0.00        1.62        2/2           spicer_ [3]
```

```
                0.00        0.01        1/1          spicea_ [23]
                0.00        0.01        1/1          spicec_ [30]
                0.00        0.00        1/1          extime_ [59]
                0.00        0.00        2/2          outres_ [83]
                0.00        0.00        1/1          spicej_ [89]
                0.00        0.00        1/1          otpfil_ [109]
                0.00        0.00        1/1          spicem_ [110]
                0.00        0.00        1/1          getnum_ [108]
-----------------------------------------------------------------
                                                     <spontaneous>
[2]     100.0   0.00        1.64                   main [2]
                0.00        1.64        1/1          sppac_ [1]
-----------------------------------------------------------------
                0.00        1.62        2/2          sppac_ [1]
[3]      98.7   0.00        1.62        2          spicer_ [3]
                0.00        1.54        1/1          spptra_ [4]
                0.00        0.07        3/3          sppdca_ [13]
                0.00        0.00        2/2          sppres_ [55]
                0.00        0.00        1/1          sppbpt_ [73]
                0.00        0.00        1/35         spmg08_ [40]
                0.00        0.00        1/1782       spmptr_ [20]
                0.00        0.00        4/28         sppxsg_ [97]
                0.00        0.00        4/28         sppxtp_ [98]
                0.00        0.00        2/8          sppior_ [100]
                0.00        0.00        1/1          sppchs_ [116]
-----------------------------------------------------------------
                0.00        1.54        1/1          spicer_ [3]
[4]      94.4   0.00        1.54        1          spptra_ [4]
                0.01        1.50      228/228        spptrx_ [5]
                0.00        0.01     1145/1185       spmrmp_ [22]
                0.00        0.01       28/303        spptct_ [11]
                0.00        0.00      229/240        sppstr_ [35]
                0.00        0.00       79/315        spmext_ [27]
                0.00        0.00        5/35         spmg08_ [40]
                0.00        0.00        8/37         spmclr_ [65]
                0.00        0.00       10/1782       spmptr_ [20]
                0.00        0.00        1/7          spmcrn_ [66]
                0.00        0.00       29/52418      spptst_ [16]
                0.00        0.00        2/4477       spptim_ [39]
                0.00        0.00      232/1153       spxc08_ [94]
-----------------------------------------------------------------
                0.01        1.50      228/228        spptra_ [4]
[5]      92.2   0.01        1.50      228          spptrx_ [5]
                0.01        1.41      289/302        sppitr_ [6]
                0.01        0.07      275/303        spptct_ [11]
                0.00        0.00      289/292        sppsor_ [37]
                0.00        0.00      289/303        sppcje_ [45]
                0.00        0.00      289/289        sppccf_ [58]
                0.00        0.00     564/52418       spptst_ [16]
                0.00        0.00        2/1153       spxc08_ [94]
-----------------------------------------------------------------
                0.00        0.06       13/302        sppdca_ [13]
                0.01        1.41      289/302        spptrx_ [5]
[6]      90.1   0.01        1.47      302          sppitr_ [6]
                0.01        1.38     1185/1185       sppldm_ [7]
                0.05        0.00      883/883        sppdcs_ [14]
                0.01        0.03      883/883        sppdcd_ [18]
                0.00        0.00       36/129        spmsiz_ [56]
```

```
                0.00        0.00     883/1153        spxc08_ [94]
-------------------------------------------------------------------
                0.01        1.38    1185/1185        sppitr_ [6]
[7]      84.6   0.01        1.38    1185         sppldm_ [7]
                0.93        0.43    1185/1186        sppmos_ [8]
                0.00        0.00    2370/2370        sppldu_ [46]
                0.00        0.00    2370/4477        spptim_ [39]
                0.00        0.00    1185/1185        sppdsr_ [52]
                0.00        0.00    1185/1239        spxz08_ [53]
                0.00        0.00    1185/52418       spptst_ [16]
                0.00        0.00      35/129         spmsiz_ [56]
                0.00        0.00    1185/1186        sppjdd_ [93]
                0.00        0.00    1185/1186        sppbjt_ [91]
                0.00        0.00    1185/1186        sppfet_ [92]
-------------------------------------------------------------------
                0.00        0.00       1/1186        sppdca_ [13]
                0.93        0.43    1185/1186        sppldm_ [7]
[8]      83.4   0.93        0.44    1186         sppmos_ [8]
                0.21        0.00    8096/8096        sppmq3_ [9]
                0.08        0.03   42156/42156       sppitg_ [10]
                0.04        0.00    8056/8056        sppflm_ [15]
                0.04        0.00    7682/7682        sppmyr_ [17]
                0.02        0.00    8056/8056        sppvds_ [19]
                0.01        0.00    8056/8056        sppjlm_ [31]
-------------------------------------------------------------------
                0.21        0.00    8096/8096        sppmos_ [8]
[9]      12.9   0.21        0.00    8096         sppmq3_ [9]
-------------------------------------------------------------------
                0.08        0.03   42156/42156       sppmos_ [8]
[10]      6.8   0.08        0.03   42156         sppitg_ [10]
                0.02        0.01   42156/52418       spptst_ [16]
-------------------------------------------------------------------
                0.00        0.01      28/303         spptra_ [4]
                0.01        0.07     275/303         spptrx_ [5]
[11]      5.4   0.01        0.08     303          spptct_ [11]
                0.08        0.01    7272/7272        sppter_ [12]
                0.00        0.00     303/52418       spptst_ [16]
-------------------------------------------------------------------
                0.08        0.01    7272/7272        spptct_ [11]
[12]      5.0   0.08        0.01    7272         sppter_ [12]
                0.00        0.00    7272/52418       spptst_ [16]
-------------------------------------------------------------------
                0.00        0.07       3/3           spicer_ [3]
[13]      4.2   0.00        0.07       3         sppdca_ [13]
                0.00        0.06      13/302         sppitr_ [6]
                0.00        0.00      15/35          spmg08_ [40]
                0.00        0.00       1/1186        sppmos_ [8]
                0.00        0.00      40/1185        spmrmp_ [22]
                0.00        0.00      12/315         spmext_ [27]
                0.00        0.00      11/240         sppstr_ [35]
                0.00        0.00       3/7           spmcrn_ [66]
                0.00        0.00      10/37          spmclr_ [65]
                0.00        0.00      14/1782        spmptr_ [20]
                0.00        0.00      12/303         sppcje_ [45]
                0.00        0.00       3/8           spmslp_ [71]
                0.00        0.00       3/292         sppsor_ [37]
                0.00        0.00       6/1239        spxz08_ [53]
                0.00        0.00       6/4477        spptim_ [39]
```

```
                0.00        0.00        3/3          spma08_ [87]
                0.00        0.00        2/1153       spxc08_ [94]
                0.00        0.00        1/1186       sppjdd_ [93]
                0.00        0.00        1/1186       sppbjt_ [91]
                0.00        0.00        1/1186       sppfet_ [92]
                0.00        0.00        1/1239       spxchm_ [90]
-----------------------------------------------------------------
                0.05        0.00      883/883        sppitr_ [6]
[14]     2.9    0.05        0.00      883          sppdcs_ [14]
                0.00        0.00      883/8055       sppixx_ [24]
-----------------------------------------------------------------
                0.04        0.00     8056/8056       sppmos_ [8]
[15]     2.7    0.04        0.00     8056         sppflm_ [15]
-----------------------------------------------------------------
                0.00        0.00        2/52418      sppvml_ [79]
                0.00        0.00        2/52418      sppppr_ [61]
                0.00        0.00        3/52418      sppbrt_ [74]
                0.00        0.00        4/52418      spmrel_ [80]
                0.00        0.00        5/52418      sppovn_ [88]
                0.00        0.00        5/52418      sppodf_ [72]
                0.00        0.00        8/52418      spplrf_ [77]
                0.00        0.00        8/52418      sppfnn_ [78]
                0.00        0.00       29/52418      spptra_ [4]
                0.00        0.00       37/52418      spmclr_ [65]
                0.00        0.00       41/52418      sppres_ [55]
                0.00        0.00       68/52418      spmgmx_ [29]
                0.00        0.00      139/52418      spmchk_ [33]
                0.00        0.00      240/52418      sppstr_ [35]
                0.00        0.00      303/52418      spptct_ [11]
                0.00        0.00      347/52418      spmext_ [27]
                0.00        0.00      564/52418      spptrx_ [5]
                0.00        0.00     1185/52418      sppldm_ [7]
                0.00        0.00     7272/52418      sppter_ [12]
                0.02        0.01    42156/52418      sppitg_ [10]
[16]     2.6    0.03        0.01    52418        spptst_ [16]
                0.01        0.00    52418/52430      sppand_ [25]
-----------------------------------------------------------------
                0.04        0.00     7682/7682       sppmos_ [8]
[17]     2.5    0.04        0.00     7682         sppmyr_ [17]
-----------------------------------------------------------------
                0.01        0.03      883/883        sppitr_ [6]
[18]     2.1    0.01        0.03      883          sppdcd_ [18]
                0.02        0.00     6181/6181       spp_dcd_ [21]
                0.01        0.00     6245/8055       sppixx_ [24]
                0.00        0.00     1766/4477       spptim_ [39]
                0.00        0.00        4/5          sppmtl_ [54]
                0.00        0.00        3/5          sppswp_ [104]
-----------------------------------------------------------------
                0.02        0.00     8056/8056       sppmos_ [8]
[19]     1.2    0.02        0.00     8056         sppvds_ [19]
-----------------------------------------------------------------
                0.00        0.00        1/1782       spicea_ [23]
                0.00        0.00        1/1782       spicer_ [3]
                0.00        0.00        1/1782       spperk_ [41]
                0.00        0.00        1/1782       sppitp_ [86]
                0.00        0.00        2/1782       spmrel_ [80]
                0.00        0.00        4/1782       sppset_ [32]
                0.00        0.00        6/1782       spprsv_ [43]
```

```
                0.00        0.00        8/1782          spmslp_ [71]
                0.00        0.00       10/1782          spptra_ [4]
                0.00        0.00       14/1782          sppdca_ [13]
                0.00        0.00       37/1782          spmclr_ [65]
                0.00        0.00       68/1782          spmgmx_ [29]
                0.00        0.00      129/1782          spmsiz_ [56]
                0.00        0.00      315/1782          spmext_ [27]
                0.01        0.00     1185/1782          spmrmp_ [22]
[20]     1.2    0.02        0.00     1782          spmptr_ [20]
                0.00        0.00     1782/3181          spmloc_ [36]
```

A.5 The results from "test-osc-1"

The results of the profile for "test-osc-1" circuit are as follows:

```
granularity: each sample hit covers 4 byte(s) for 0.01% of 7.33 seconds

                                          called/total       parents
index  %time    self descendents  called+self    name     index
                                          called/total       children

                0.00        7.33        1/1             main [2]
[1]     100.0   0.00        7.33        1           sppac_ [1]
                0.00        7.30        2/2             spicer_ [3]
                0.00        0.01        1/1             spicea_ [40]
                0.00        0.01        1/1             spicec_ [43]
                0.00        0.00        2/2             outres_ [52]
                0.00        0.00        3/3             spicex_ [85]
                0.00        0.00        1/1             spiceb_ [88]
                0.00        0.00        3/3             spicej_ [91]
                0.00        0.00        1/1             extime_ [96]
                0.00        0.00        1/1             otpfil_ [126]
                0.00        0.00        1/1             spicem_ [127]
                0.00        0.00        1/1             getnum_ [125]
-----------------------------------------------
                                                    <spontaneous>
[2]     100.0   0.00        7.33                main [2]
                0.00        7.33        1/1             sppac_ [1]
-----------------------------------------------
                0.00        7.30        2/2             sppac_ [1]
[3]      99.6   0.00        7.30        2           spicer_ [3]
                0.13        7.16        2/2             spptra_ [4]
                0.00        0.01        4/4             sppdca_ [51]
                0.00        0.00        2/2             sppres_ [59]
                0.00        0.00        1/1             sppbpt_ [67]
                0.00        0.00        1/44            spmg08_ [53]
                0.00        0.00        1/47222         spmptr_ [11]
                0.00        0.00        4/46            sppxsg_ [103]
                0.00        0.00        4/46            sppxtp_ [104]
                0.00        0.00        2/2             sppchs_ [116]
                0.00        0.00        2/2             sppior_ [119]
-----------------------------------------------
                0.13        7.16        2/2             spicer_ [3]
[4]      99.4   0.13        7.16        2           spptra_ [4]
```

```
                0.03        6.37     4610/4610          spptrx_ [5]
                0.01        0.37    23060/23060         spmrmp_ [16]
                0.01        0.13     9165/13935         spmsiz_ [20]
                0.02        0.09     5018/5482          spmext_ [24]
                0.01        0.08     4553/4555          spmrel_ [27]
                0.03        0.00     4610/4610          sppstr_ [34]
                0.02        0.00     9175/19950         spxc08_ [30]
                0.00        0.00       10/44            spmg08_ [53]
                0.00        0.00       12/4630          spptct_ [15]
                0.00        0.00       15/45            spmclr_ [69]
                0.00        0.00       20/47222         spmptr_ [11]
                0.00        0.00        2/9             spmcrn_ [76]
                0.00        0.00       19/177540        spptst_ [22]
                0.00        0.00        4/52290         spptim_ [32]
-----------------------------------------------------------------------
                0.03        6.37     4610/4610          spptra_ [4]
[5]      87.2   0.03        6.37     4610           spptrx_ [5]
                0.22        5.51     4622/4626          sppitr_ [6]
                0.05        0.36     4618/4630          spptct_ [15]
                0.14        0.07     4622/4626          sppsor_ [19]
                0.01        0.00     4622/4622          sppccf_ [46]
                0.01        0.00     9240/177540        spptst_ [22]
                0.00        0.00        4/19950         spxc08_ [30]
-----------------------------------------------------------------------
                0.00        0.00        4/4626          sppdca_ [51]
                0.22        5.51     4622/4626          spptrx_ [5]
[6]      78.2   0.22        5.51     4626           sppitr_ [6]
                0.36        2.38    15370/15370         sppldm_ [7]
                1.77        0.02    10744/10744         sppdcs_ [8]
                0.12        0.83    10744/10744         sppdcd_ [10]
                0.03        0.00    10744/19950         spxc08_ [30]
                0.00        0.00        2/13935         spmsiz_ [20]
-----------------------------------------------------------------------
                0.36        2.38    15370/15370         sppitr_ [6]
[7]      37.4   0.36        2.38    15370           sppldm_ [7]
                1.08        0.19    15370/15372         sppbjt_ [9]
                0.52        0.00   276720/307450        sppldu_ [12]
                0.10        0.09    15362/15362         sppldc_ [21]
                0.10        0.07    15370/15370         sppldl_ [23]
                0.01        0.09    15370/15406         spxz08_ [25]
                0.05        0.00    15370/15370         sppdsr_ [29]
                0.02        0.00    30740/52290         spptim_ [32]
                0.02        0.00    15370/15372         sppjdd_ [36]
                0.01        0.00    15370/177540        spptst_ [22]
                0.01        0.00    15370/15372         sppmos_ [41]
                0.01        0.00    15370/15372         sppfet_ [47]
                0.00        0.00       14/13935         spmsiz_ [20]
-----------------------------------------------------------------------
                1.77        0.02    10744/10744         sppitr_ [6]
[8]      24.4   1.77        0.02    10744           sppdcs_ [8]
                0.02        0.00    10744/270128        sppixx_ [14]
-----------------------------------------------------------------------
                0.00        0.00        2/15372         sppdca_ [51]
                1.08        0.19    15370/15372         sppldm_ [7]
[9]      17.3   1.08        0.19    15372           sppbjt_ [9]
                0.10        0.06    59256/89976         sppitg_ [18]
                0.03        0.00    28540/28540         sppjlm_ [35]
-----------------------------------------------------------------------
```

```
                0.12        0.83    10744/10744        sppitr_ [6]
[10]    13.0    0.12        0.83    10744          sppdcd_ [10]
                0.42        0.00    257856/257856      spp_dcd_ [13]
                0.40        0.00    258826/270128      sppixx_ [14]
                0.01        0.00    21488/52290        spptim_ [32]
                0.00        0.00      42/51            sppswp_ [64]
                0.00        0.00       4/5             sppmtl_ [73]
-----------------------------------------------------------------
                0.00        0.00       1/47222         spicea_ [40]
                0.00        0.00       1/47222         spicer_ [3]
                0.00        0.00       1/47222         spperk_ [60]
                0.00        0.00       2/47222         sppitp_ [62]
                0.00        0.00       4/47222         sppset_ [44]
                0.00        0.00       9/47222         spmslp_ [77]
                0.00        0.00      14/47222         spprsv_ [49]
                0.00        0.00      16/47222         sppdca_ [51]
                0.00        0.00      20/47222         spptra_ [4]
                0.00        0.00      45/47222         spmclr_ [69]
                0.00        0.00      77/47222         spmgmx_ [50]
                0.06        0.00    4555/47222         spmrel_ [27]
                0.07        0.00    5482/47222         spmext_ [24]
                0.18        0.01   13935/47222         spmsiz_ [20]
                0.30        0.02   23060/47222         spmrmp_ [16]
[11]    9.0     0.62        0.04   47222           spmptr_ [11]
                0.03        0.01   47222/70552         spmloc_ [28]
-----------------------------------------------------------------
                0.03        0.00   15360/307450        sppldc_ [21]
                0.03        0.00   15370/307450        sppldl_ [23]
                0.52        0.00   276720/307450       sppldm_ [7]
[12]    7.9     0.58        0.00   307450          sppldu_ [12]
-----------------------------------------------------------------
                0.42        0.00   257856/257856       sppdcd_ [10]
[13]    5.7     0.42        0.00   257856          spp_dcd_ [13]
                0.00        0.00     156/156           spprsx_ [65]
                0.00        0.00      13/5482          spmext_ [24]
-----------------------------------------------------------------
                0.00        0.00       8/270128        sppset_ [44]
                0.00        0.00     550/270128        sppmtl_ [73]
                0.02        0.00   10744/270128        sppdcs_ [8]
                0.40        0.00   258826/270128       sppdcd_ [10]
[14]    5.6     0.41        0.00   270128          sppixx_ [14]
-----------------------------------------------------------------
                0.00        0.00      12/4630          spptra_ [4]
                0.05        0.36    4618/4630          spptrx_ [5]
[15]    5.6     0.05        0.36    4630            spptct_ [15]
                0.33        0.02   23150/23150         sppter_ [17]
                0.00        0.00    4630/177540        spptst_ [22]
-----------------------------------------------------------------
                0.01        0.37   23060/23060          spptra_ [4]
[16]    5.1     0.01        0.37   23060            spmrmp_ [16]
                0.30        0.02   23060/47222          spmptr_ [11]
                0.01        0.01   23060/27796          spmadj_ [33]
                0.01        0.00   23060/70552          spmloc_ [28]
-----------------------------------------------------------------
                0.33        0.02   23150/23150          spptct_ [15]
[17]    4.9     0.33        0.02   23150            sppter_ [17]
                0.02        0.01   23150/177540         spptst_ [22]
-----------------------------------------------------------------
```

```
                0.03    0.02   15360/89976       sppldc_ [21]
                0.03    0.02   15360/89976       sppldl_ [23]
                0.10    0.06   59256/89976       sppbjt_ [9]
[18]     3.3    0.15    0.09   89976         sppitg_ [18]
                0.07    0.02   89976/177540      spptst_ [22]
-----------------------------------------------------------
                0.00    0.00       4/4626        sppdca_ [51]
                0.14    0.07    4622/4626        spptrx_ [5]
[19]     2.9    0.14    0.07    4626          sppsor_ [19]
                0.00    0.06    4622/13935       spmsiz_ [20]
-----------------------------------------------------------
                0.00    0.00       1/13935       spiceb_ [88]
                0.00    0.00       1/13935       sppvml_ [86]
                0.00    0.00       2/13935       spicea_ [40]
                0.00    0.00       2/13935       sppvar_ [83]
                0.00    0.00       2/13935       sppitr_ [6]
                0.00    0.00       4/13935       sppbrt_ [68]
                0.00    0.00       5/13935       spperk_ [60]
                0.00    0.00      14/13935       sppldm_ [7]
                0.00    0.00      17/13935       sppfnd_ [75]
                0.00    0.00      26/13935       spprsx_ [65]
                0.00    0.00      74/13935       spprsv_ [49]
                0.00    0.06    4622/13935       sppsor_ [19]
                0.01    0.13    9165/13935       spptra_ [4]
[20]     2.8    0.01    0.19   13935          spmsiz_ [20]
                0.18    0.01   13935/47222       spmptr_ [11]
```

Appendix B

Circuit Descriptions

B.1 Circuit "test-adder"

The "test-adder" circuit description is as follows:

```
* 1-BIT ALL NAND GATE BINARY ADDER *
X1 1 2 3 0 5 6 ADDER1b
VCC 6 0 5V
VIN1A 1 0 PULSE(0 3 0 10NS 10NS 20NS 60NS)
VIN1B 2 0 PULSE(0 3 60NS 10NS 10NS 50NS 120NS)
R0 3 0 1K
R1 5 0 1K
.TR 5NS 150NS
.PRINT TR V(1) V(2) V(3) V(5)
.SUBCKT NAND2 1 2 3 4
Q1 9 5 1 QMOD
D1C 0 1 DMOD
Q2 9 5 2 QMOD
D2C 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDR 10 3 DMOD
Q5 3 8 0 QMOD
.MODEL DMOD D
.MODEL QMOD NPN(BF=74 RB=100 CJE=1PF CJC=3PF)
.ENDS
.SUBCKT ADDER1b 1 2 3 4 5 6
X1 1 2 7 6 NAND2
X2 1 7 8 6 NAND2
X3 2 7 9 6 NAND2
X4 8 9 10 6 NAND2
X5 4 10 11 6 NAND2
X6 4 11 12 6 NAND2
X7 10 11 13 6 NAND2
X8 12 13 3 6 NAND2
X9 11 7 5 6 NAND2
.ENDS
.END
```

B.2 Circuit "test-adder-2"

The "test-adder-2" circuit description is as follows:

```
* 2-BIT ALL NAND GATE BINARY ADDER *
X1 1 2 3 0 5 6 ADDER1b
X2 7 8 9 5 10 6 ADDER1b
VCC 6 0 5V
VIN1A 1 0 PULSE(0 3 0 10NS 10NS 20NS 60NS)
VIN1B 2 0 PULSE(0 3 60NS 10NS 10NS 50NS 120NS)
VIN2A 7 0 PULSE(0 3 120NS 10NS 10NS 110NS 240NS)
VIN2B 8 0 PULSE(0 3 240NS 10NS 10NS 230NS 480NS)
R0 3 0 1K
R1 10 0 1K
.TR 5NS 500NS
.PRINT TR V(1) V(2) V(3) V(7) V(8) V(9)
.OPTION ITL5=0
.SUBCKT NAND2 1 2 3 4
Q1 9 5 1 QMOD
D1C 0 1 DMOD
Q2 9 5 2 QMOD
D2C 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDR 10 3 DMOD
Q5 3 8 0 QMOD
.MODEL DMOD D
.MODEL QMOD NPN(BF=74 RB=100 CJE=1PF CJC=3PF)
.ENDS
.SUBCKT ADDER1b 1 2 3 4 5 6
X1 1 2 7 6 NAND2
X2 1 7 8 6 NAND2
X3 2 7 9 6 NAND2
X4 8 9 10 6 NAND2
X5 4 10 11 6 NAND2
X6 4 11 12 6 NAND2
X7 10 11 13 6 NAND2
X8 12 13 3 6 NAND2
X9 11 7 5 6 NAND2
.ENDS
.END
```

B.3 Circuit "test-inv-1"

The "test-inv-1" circuit description is as follows:

```
* The design and analysis of VLSI circuits (Glasser,Dobberpuhl)
.OPTIONS LIMPTS=501 DEFL=2.25E-6
VIN 1 0 PULSE(0 5 0 10NS 10NS 20NS 60NS)
VDD 9 0 5V
* ... MOS inverter
M1E 2 1 0 0 NENHS W=11.2U AD=61P PD=42U
.MODEL NENHS NMOS LEVEL=3 RSH=0 TOX=330E-10 LD=0.19E-6 U0=650
+       XJ=0.27E-6 VMAX=13E4 ETA=0.25 KAPPA=0.5 NSUB=5E14
```

```
+      THETA=0.1 VTO=0.946 CGSO=2.43E-10 CGDO=2.43E-10
+      CJ=6.9E-5 CJSW=3.3E-10 PB=0.7 MJ=0.5 MJSW=0.3 NFS=1E10
M1D 9 2 2 0 NDEPS W=4.2U L=6.25U
.MODEL NDEPS NMOS LEVEL=3 RSH=0 TOX=330E-10 LD=0.19E-6 UO=650
+      XJ=0.27E-6 VMAX=13E4 ETA=0.25 KAPPA=0.5 NSUB=50E14
+      THETA=0.04 VTO=-2.078 CGSO=2.43E-10 CGDO=2.43E-10
+      CJ=6.9E-5 CJSW=3.3E-10 PB=0.7 MJ=0.5 MJSW=0.3 NFS=1E10
* ... MOS inverter
M2E 3 2 0 0 NENHS W=11.2U AD=61P PD=42U
M2D 9 3 3 0 NDEPS W=4.2U L=6.25U
M3E 4 3 0 0 NENHS W=11.2U AD=61P PD=42U
M3D 9 4 4 0 NDEPS W=4.2U L=6.25U
M4E 5 4 0 0 NENHS W=11.2U AD=61P PD=42U
M4D 9 5 5 0 NDEPS W=4.2U L=6.25U
* analysis
.TRAN 5NS 200NS
.PRINT TR V(1) V(2) V(3) V(4)
.DC VIN 0 5 0.5
.PRINT DC V(2)
.END
```

B.4 Circuit "test-osc-1"

The "test-osc-1" circuit description is as follows:

```
.OPT ITL5=0  LIMPTS=1001  NOTIME
.TRAN 1NS 100NS 0 50PS UIC
.PRINT TRAN V(8) V(5,8) I(VC)
QH 4 2 0 TBH
TENTRE 2 0 1 0 Z0=50 TD=78.3PS
VB 1 0 DC 1.4523
TSORTIE 4 0 5 0 Z0=50 TD=170.3PS
TPARA 5 0 6 0 Z0=50 TD=127PS
L1 5 7 10N IC=-0.1
VC 7 0 DC 2.5
CDECC 5 8 0.1N IC=1.0
RCHARGE 8 0 50
ICOC 4 0 PULSE  0  1E-2 0 0.1N 0.1N 0.1N
.MODEL TBH NPN (IS=0.4E-24 NF=1.011 NR=1 BF=30 BR=1 ISE=0.1E-14
+  NE=2 VAF=40 IKF=0.8 VAR=25 TF=0.58E-11 PTF=65 XTF=0.5087
+  ISC=1.2E-13 NC=2 CJE=0.1387E-12 MJE=0.5 VJE=1.708 CJC=0.1403E-12
+  MJC=0.5 VJC= 1.449 XCJC=0.2857 RB=20.9 RE=7.8 RC= 36 IKR=0.8
+  EG=1.424 RBM=17 IRB=6E-3)
.END/EXT
.VAR L1'IC
.VAR CDECC'IC
.END
```