Naval Command,
Control and Ocean
Surveillance Center RDT&E Division San Diego, CA
92152-5001

Technical Report 1572
March 1993

Decomposition of Large Sparse Symmetric Systems for Parallel Computation

Part 1: Theoretical Foundations

A. K. Kevorkian

Technical Report 1572
March 1993

Decomposition of Large Sparse Symmetric Systems for Parallel Computation

Part 1: Theoretical Foundations

A. K. Kevorkian

NAVAL COMMAND, CONTROL AND
OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

This report was sponsored by the Office of the Chief of Naval Research under accession number DN302038, program element 0601152, project number ZW62.

Released by
A. K. Kevorkian
Code 7304

Under authority of
J. A. Roese, Head
Signal and Information
Processing Division

ACKNOWLEDGMENT

Dr. Michael Heath read an earlier version of this work. His constructive comments and valuable suggestions were very helpful in improving the presentation of the material to this form.

This work was funded by the Naval Command, Control and Ocean Surveillance Center RDT&E Division Independent Research Program, as well as the High Performance Computing Fellowship Program. The author gratefully acknowledges both supports.

LH

EXECUTIVE SUMMARY

OBJECTIVE

Given any linear system of equations Mx = b in which M is a large sparse symmetric matrix, provide an efficient algorithm for generating smaller computational tasks that can be processed independently of each other by the different processors of a parallel architecture computer. Such an automatic decomposition is essential for the effective applications of parallel architecture computers.

RESULTS

We have presented an efficient linear-time algorithm for decomposing a large sparse symmetric system of equations Mx = b into independently solvable smaller tasks that can be executed in parallel on different processors of a parallel architecture computer. The independent tasks generated by the algorithm include all full principal submatrices of M that preserve sparsity during the symbolic factorization stage of the solution process. A detailed computer implementation of the algorithm called roadmap is covered in part 2 (Kevorkian, 1993).

CONTENTS

FIGURES

PART 1. THEORETICAL FOUNDATIONS

1. INTRODUCTION

The solution of linear systems of equations

$$Mx = b, \tag{1}$$

in which M is a large sparse symmetric matrix, forms the most central piece in many critically important and computationally demanding applications in government and industry. In the majority of these applications, there is significant parallelism hidden in the structure of the sparse matrix M. The larger and sparser the matrix, the greater the opportunities for parallelism. But as matrices get larger and their sparsity structures come to be more irregular, the harder it becomes to exploit parallelism in the problem. The basics for the direct solution of sparse symmetric and unsymmetric systems on conventional machines are well covered in Duff, Erisman, and Reid (1986) and George and Liu (1981). For parallel architecture machines, the work in Heath, Ng, and Peyton (1991) provides a detailed and comprehensive survey of the significant progress made so far on parallel algorithms for sparse problems.

In this work, we develop an algorithmic tool for exploiting the sparsity structure of large symmetric matrices to generate computational tasks that can be processed in parallel on different processors of a parallel architecture computer. These computationally independent tasks are often referred to in the scientific literature as parallel regions of computation.

Given any sparse symmetric matrix M, we model the zero-nonzero structure of M using an undirected graph G = (V, E). With respect to the graph G, the four key components of our parallelization tool are then as follows:

1. Compute the set of vertices $S = \{v \in V | \exists\ (v,w) \in E \text{ with } deg_G v > deg_G w\}$;
2. Compute connected components of induced subgraph G(V-S);
3. Classify clique connected components of G(V-S);
4. Compute independent cliques in nonclique connected component of G(V-S).

The construction of the set of vertices S forms the most novel and critical component of the parallelization tool. To highlight the key property of the set S, consider any clique in G with vertex set U, and suppose we partition U into two disjunct parts U' and U" such that $U' = \{u \mid deg_G u \le deg_G v \text{ for all } v \in U\}$. We call the set of vertices U' the core of clique G(U), and for the special case where a vertex u in U' satisfies the equality $deg_G u = |U|-1$, we call U' the interior of G(U) and the induced subgraph G(U') an interior clique. The matrix interpretation of an interior clique makes this subgraph of G of some interest in sparse matrix computations. To highlight this, let G(U) be any interior clique in G and let A be a principal submatrix of M corresponding to G(U). Then, in subsequent developments, we show that the symbolic factorization of submatrix A does not produce fill-in. Interior cliques are thus ideally suited for sparse matrix computations since reductions in fill-in generally improve the overall efficiency of a solution process.

Going back to step 1 of the parallelization tool, we show that every interior clique in the graph G is a connected component of the induced subgraph G(V-S), and so steps 1 and 2 of the parallelization tool isolate every principal submatrix of a sparse symmetric matrix, which preserves sparsity in the process of symbolic factorization. Step 3 of the parallelization tool deals with the connected components of G(V-S) that are cliques but not interior, whereas step 4 focuses on those connected components of G(V-S) that are not cliques. The objective of step 3 is to classify the noninterior cliques in G using an interior clique as the clique of choice. The purpose of step 4 is to exploit the underlying structure of a nonclique connected component.

Through the four algorithmic steps embodied in the parallelization tool, we obtain a vertex partition

$$\Pi^* = (V_1, V_2, \ldots, V_r, S^*),$$

where

$$S \subseteq S^*,$$

and such that the following three properties are satisfied:

(a) For any two distinct elements V_i and V_j of the partition, no vertex in V_i is adjacent to a vertex in V_j;
(b) Every element V_i of the partition induces a clique in G;
(c) Interior of every clique in G is an element of the partition.

By properties (a) and (b), the graph with respect to the vertex partition Π^* is a star-shaped graph with root vertex S^* and r leaf vertices V_1 through V_r each inducing a clique in G. The leading r elements of the partition thus correspond to r dense computational tasks, which can be processed in parallel on different processors of a parallel architecture machine. In the case where the matrix M in equation (1) is positive definite, each of the r independent computational tasks requires Cholesky factorization of a full nonsingular principal submatrix of M as well as the solution of a full triangular system with multiple right-hand sides. Property (c) of the vertex partition is a means for keeping the fill-in acceptably small.

For an arbitrary sparse symmetric matrix, the problem of finding an ordering that minimizes fill-in is NP-complete (Yannakakis, 1981), and so most ordering methods must rely on heuristics to produce small fill-in. Among the many heuristic methods proposed and developed over the last 35 years, the minimum degree algorithm (George & Liu, 1981) stands out as the most popular and widely used method. The ordering scheme we have described in this work uses the degrees of vertices as well, but in a very different way. For each vertex v in the set of vertices $S \subset S^*$, there must exist in E an edge (v, w) such that $\deg_G v > \deg_G w$. The combined use of edges and degrees in step 1 of the parallelization tool makes our ordering scheme distinctly different from the minimum degree method.

This report is organized as follows. In section 2, we cover the necessary graph-theoretic notation. In section 3, we give a concise formulation of the concepts of "core" and "interior clique" and subsequently derive the sparsity preserving and parallelism properties of an interior clique. In section 4, we motivate the four-step

parallelization tool. In sections 5 through 8, we give detailed analysis and high-level implementations of steps 1 through 4 of the parallelization tool. In section 9, we show that the parallelization tool has linear-time complexity. A computer implementation of this parallelization tool called "roadmap" is covered in Kevorkian (1993). The computer program roadmap was developed using the widely available linear algebra package Matlab (MathWorks, 1990). In section 10, we discuss matrix interpretations of vertex partitions produced by the parallelization tool. In particular, we show that the problem of constructing a vertex partition Π^* satisfying property (a) is equivalent to computing a permutation matrix P such that PMP^T is an (r+1)-by-(r+1) block bordered diagonal matrix. The use of block bordered diagonal forms has recently become an area of active research in diverse disciplines of science and engineering (Zhang, Byrd & Schnabel, 1992). A main motivation has been the exploitation of parallelism in scientific computing. Section 10 gives detailed background on the methodologies adopted to date for permuting symmetric matrices into block bordered diagonal form. In section 11, we discuss a strategy for solving sparse positive definite systems of equations using vertex partitions produced by the parallelization tool. In section 12, we discuss the exploitation of parallelism in large Schur complements and our progress in the development of a recursive parallelization tool. In section 13, we address issues concerning the proper distribution of workload across processors. In section 14, we illustrate the application of the computer program roadmap to a sparse symmetric matrix with irregular sparsity structure.

2. NOTATION

A graph G = (V, E) consists of a finite, nonempty set of vertices V and a set of edges E. If the edges are ordered pairs (u, v) of vertices, G is said to be directed. If the edges are unordered pairs of vertices, also denoted by (u, v), G is said to be undirected. All graphs in this work are assumed to be undirected and connected. For a subset U of the vertex set V, the induced subgraph G(U) of G is the subgraph G(U) = (U, E(U)) where

$$E(U) = \{(u,v) \in E | u, v \in U\}.$$

A set of vertices S is called a separator of G if the induced subgraph G(V-S) is disconnected.

In a graph G = (V, E), a vertex v is said to be adjacent to another vertex w if the pair (v, w) is an edge in E. The set

$$adj_G v = \{w \in V - \{v\} | (v, w) \in E\}$$

denotes the set of vertices adjacent to v. The degree of vertex v, denoted by $deg_G v$, is the number of vertices adjacent to v and so we have

$$deg_G v = |adj_G v|.$$

A graph G = (V, E) is said to be regular if all its vertices have the same degree. An induced subgraph G(U) of G is called a clique if each vertex in U is adjacent to every other vertex in U. A clique is maximal if it is not a proper subgraph of another clique. For any graph G = (V, E), a vertex partition is called a clique partition if each element of the partition induces a clique. A vertex v in G is called simplicial (Lekkerkerker & Boland, 1962; Dirac, 1961) if the subgraph of G induced by the vertex set $adj_G v$ is a clique.

For any graph G = (V, E), the complement of G is the graph $G^* = (V, E^*)$ with edge set E^* defined by

$$E^* = \{(v, w) | v, w \in V \text{ and } (v, w) \notin E\}.$$

Thus for any two distinct vertices v and w, the pair (v, w) is an edge in E^* if and only if (v, w) is not an edge in E. This means that for any subset U of V, the graph

$$G' = (U, E(U) \cup E^*(U))$$

is a clique.

For every n-by-n structurally symmetric matrix $M = [m_{ij}]$, there exists an undirected graph G = (V, E) such that vertex v_i in V represents row i of M and the edge (v_i, v_j) is in E if and only if $m_{ij} \neq 0$ for all $i \neq j$.

Another interesting representation for a graph G = (V, E) is by means of vertex partitions. For any vertex partition $\Pi = (V_1, V_2, \ldots, V_k)$ *in* G, $G_\Pi = (V_\Pi, E_\Pi)$ is a graph such that each element V_i of the partition is a vertex in V_Π and the pair (V_i, V_j) is an edge in E_Π if and only if a vertex in the set V_i is adjacent to a vertex in the set V_j. George and Liu (1981) call G_Π a quotient graph of G with respect to Π. Quotient graphs are well suited for exploiting the underlying structure of block matrices.

3. CORE OF A CLIQUE AND INTERIOR CLIQUES

The central idea in this work concerns the partitioning of the set of vertices in an arbitrary clique into two disjunct parts using the degrees of the vertices in the clique.

For a clique G(U) in the graph G = (V, E), the core of G(U) is the set of vertices cor(U) defined by

$$cor(U) = \{u \in U | deg_G u = \min_{v \in U} deg_G v\}.$$

The set cor(U) partitions the vertices in clique G(U) into two disjunct parts cor(U) and U - cor(U) (possible empty) satisfying the following two properties:

(a) $deg_G u \geq |U| - 1$ for any $u \in cor(U)$,

(b) $deg_G u > |U| - 1$ for any $u \in U - cor(U)$.

Statement (a) holds since $\deg_G u \geq \deg_{G(U)} v = |U| - 1$ for any $v \in U$. Statement (b) is obtained by contradiction. Let u be any vertex in U - cor(U) such that $\deg_G u = |U| - 1$. Then $\deg_G u \leq \deg_G v$ for any $v \in U$ and so u must be in cor(U), and thus we have a contradiction.

The single difference between the conditions in statements (a) and (b) is the equality case $\deg_G u = |U| - 1$ in statement (a). This equality associates with the set of vertices cor(U) properties that will prove very useful from the standpoint of exploiting parallelism and preserving sparsity in sparse undirected graphs. To facilitate our handling of this particular instance, we introduce the following special case of the core of a clique.

For a clique G(U) in G = (V, E), the interior of G(U) is the set of vertices int(U) defined by

$$\text{int}(U) = \{u \in U \mid \deg_G u = |U| - 1\}.$$

Also, the exterior of the clique G(U) is the vertex set ext(U) defined by

$$\text{ext}(U) = U - \text{int}(U).$$

From the definitions of int(U) and cor(U), it is easy to see that for any clique G(U) in G the following relation holds

$$\text{cor}(U) = \text{int}(U)$$

if and only if

$$\min_{v \in U} \deg_G v = |U| - 1.$$

Therefore, the core and interior of a clique G(U) are identical if and only if there exists in U a vertex v such that $\deg_G v = |U| - 1$. Otherwise, G(U) is a clique with an empty interior. In the case that G(U) is a clique with a nonempty interior, we call the clique induced by the vertex set int(U) an interior clique.

Before we present the main results in this work, we will first explore the role of interior cliques in sparse matrix computations and parallel processing and also establish connections to other works whenever appropriate.

Let G(U) be any clique in G = (V, E). Then for any vertex u in U the following two conditions are equivalent:

(a) $\deg_G u = |U| - 1$.
(b) $\text{adj}_G u = U - \{u\}$.

By condition (b), we have $\{u\} \cup \text{adj}_G u = U$, and so the vertex u together with the vertices that are adjacent to u form a clique, which means that every vertex in an interior clique is a simplicial vertex. This connection between an interior clique and a simplicial vertex provides a number of worthwhile facts, which are summarized in the next result.

Lemma 1. Let G(U) be any clique with a nonempty interior in G = (V, E). Then the following statements are true:

(a) G(U) is a maximal clique.
(b) No vertex in int(U) is adjacent to a vertex in V - U.
(c) No vertex in U is contained in the interior of any other clique in G.

Proof. Statements (a) and (b): Let u be any vertex in int(U). Then we have $adj_G u = U - \{u\}$, which means that u is not adjacent to any vertex in V - U. This completes the proof of statements (a) and (b). Statement (c): The proof is by contradiction. Let G(U') be any clique in G with $U' \neq U$. If the interior of G(U') is empty, we have nothing to prove. So Let G(U') be any clique in G with nonempty interior. Assume for contradiction that the set $int(U) \cap int(U')$ is nonempty, and let v be any vertex in $int(U) \cap int(U')$. Since vertex v is in both int(U) and int(U'), we have $adj_G v = U - \{v\}$ and $adj_G v = U' - \{v\}$ which means that U = U' and so we have a contradiction. Thus for any two distinct cliques G(U) and G(U') with nonempty interiors, the following relation must hold

$$int(U) \cap int(U') = \varnothing. \tag{2}$$

Now assume for contradiction that the intersection $ext(U) \cap int(U')$ is nonempty, and let v be any vertex in the set $ext(U) \cap int(U')$. Since v is a vertex in int(U'), we have $adj_G v = U' - \{v\}$. Also, since G(U) is a clique and v is a vertex in ext(U), we have $U - \{v\} \subseteq adj_G v$. Thus $U \subseteq U'$ and so we obtain $U \subset U'$ since $U \neq U'$. But by statement (a) the set U cannot be a proper subset of U' since G(U) is a maximal clique and G(U') is a clique. Thus we have a contradiction, and so for any two distinct cliques G(U) and G(U') with nonempty interiors, the following relation must also hold

$$ext(U) \cap int(U') = \varnothing. \tag{3}$$

The proof of statement (c) is now completed by first combining relations (2) and (3) and then making use of the equality $U = int(U) \cup ext(U)$.

3.1. PARALLELISM PROPERTY OF INTERIOR CLIQUES

Lemma 1 provides the key connection between interior cliques and parallel computations on sparse symmetric matrices. To highlight this, let G = (V, E) be any graph and let U be any proper subset of V. Assume that G(U) is a clique with a nonempty interior. By construction, the sets int(U) and ext(U) form a partition of U, and so the triple defined by

$$\Pi = (int(U), V\text{-}U, ext(U))$$

is a vertex partition in G.

Let $G_\Pi = (V_\Pi, E_\Pi)$ be the graph with respect to the vertex partition Π. By statement (b) of Lemma 1, no vertex in int(U) is adjacent to any vertex in V-U. Thus no edge in E_Π connects the two vertices int(U) and V-U in V_Π and so the graph G_Π takes the star-shaped form shown in figure 1 since G is a connected graph.

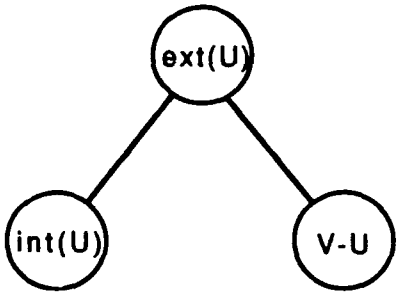

Figure 1. The graph G_Π with respect to vertex partition Π = (int(U), V-U, ext(U)).

The graph G_Π in figure 1 identifies the set of vertices ext(U) as a separator of the graph G. This means that the interior clique G(int(U)) must be a connected component of the induced subgraph G(V-ext(U)). This interpretation of G_Π together with statement (c) of Lemma 1 provides a unified framework for gaining insight into the underlying structure of a sparse undirected graph.

So let us pick in the graph G = (V, E) any two distinct cliques G(U) and G(U') with nonempty interiors and use the four disjunct vertex sets int(U), int(U'), V - U - U', and ext(U) $\cup$ ext(U') to form a vertex partition Π defined by

$$\Pi = (\text{int}(U), \text{int}(U'), V - U - U', \text{ext}(U) \cup \text{ext}(U')) .$$

Our next result sheds insight into the structure of the graph G_Π with respect to the vertex partition Π and thereby establishes the key "parallelism" property of interior cliques in general sparse undirected graphs.

Theorem 1. Let G(U) and G(U') be any two distinct cliques in G = (V, E) with nonempty interiors. Then the interior cliques G(int(U)) and G(int(U')) are connected components of the induced subgraph G(V - ext(U) – ext(U')).

Proof. Let V' = V - ext(U) - ext(U'), and assume for contradiction that G(int(U)) is not a connected component of G(V'). Then one of the following three conditions must hold:

(a) int(U) $\cap$ int(U') $\neq \emptyset$,
(b) for some vertex u in int(U), u is adjacent to a vertex in the set V - U - U',
(c) for some vertex u in int(U), u is adjacent to a vertex in int(U'),

since V' = int(U) $\cup$ int(U') $\cup$ (V - U - U').

By relation (2), condition (a) cannot hold. By statement (b) of Lemma 1, condition (b) cannot hold. Thus condition (c) must hold, and so there exists in E an edge (u, w) such that w is in int(U'). This means that w is a vertex in ext(U) since by statement (b) of Lemma 1 any vertex in V - int(U) adjacent to u must be in ext(U).

Thus we have $ext(U) \cap int(U') \neq \emptyset$ since w is in both ext(U) and int(U'), and so by relation (3) we have a contradiction. This completes the proof.

As an immediate consequence of Theorem 1, we obtain the following corollary pertaining to the parallelism property of interior cliques.

Corollary 1.1. Let $G(U_1)$ through $G(U_k)$ be any k distinct cliques in G = (V, E) with nonempty interiors, and let S be the subset of V defined by

$$S = \bigcup_{i=1}^{k} ext(U_i).$$

Then the k interior cliques $G(int(U_1))$ through $G(int(U_k))$ are connected components of the induced subgraph G(V-S).

Proof. The proof is obtained by repeated application of Theorem 1.

Thus if we let Π be a vertex partition in G = (V, E) defined by

$$\Pi = (int(U_1), int(U_2), \ldots, int(U_k), V - \bigcup_{i=1}^{k} U_i, \bigcup_{i=1}^{k} ext(U_i)), \qquad (4)$$

then by Corollary 1.1 it follows that the graph G_Π with respect to the vertex partition Π has the star-shaped form shown in figure 2. This form of the original graph G clearly exhibits the parallelism property of any interior clique in G .

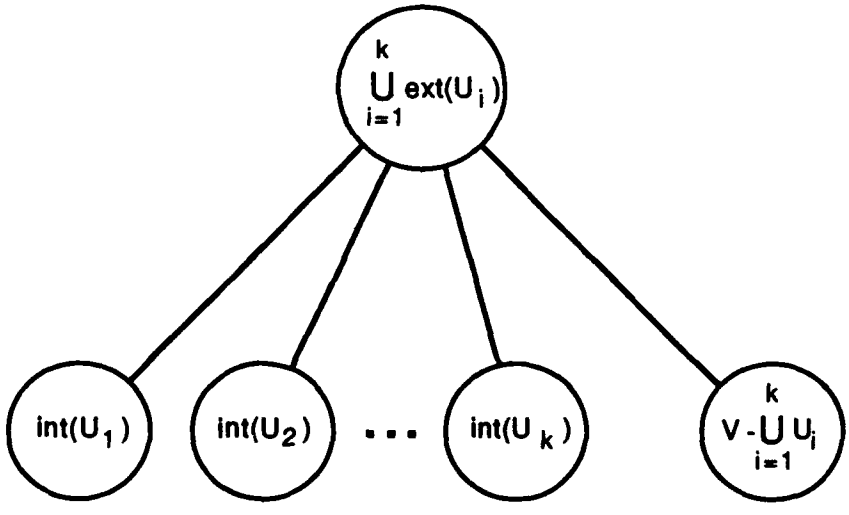

Figure 2. The graph G_Π with respect to the vertex partition Π in relation (4).

3.2 SPARSITY-PRESERVING PROPERTY OF AN INTERIOR CLIQUE

Let $Mx = b$ be any system of linear equations in which M is an n-by-n symmetric matrix with nonzero diagonal, and suppose we use the ith equation in the system $Mx = b$ to eliminate the ith component of x from the remaining n -1 equations. This numerical process of transforming the original system of n equations into a reduced system of n -1 equations has a concise graph-theoretic interpretation due to Parter (1961).

Suppose G = (V, E) is the undirected graph of the n-by-n symmetric matrix M and let v denote the vertex in V representing the ith row of M. Then the set of edges defined by

$$\mathrm{def}_G v = \{ (u,w) \mid u, w \in \mathrm{adj}_G v, (u,w) \notin E, u \neq w \}$$

correspond exactly to the fill-in produced when the ith component of x is eliminated from the original system (assuming no cancellation of nonzero elements). The set of edges $\mathrm{def}_G v$ is called the deficiency of v in G (Rose, Tarjan, & Lueker, 1976). The graph

$$G_v = (V - \{v\}, E(V - \{v\}) \cup \mathrm{def}_G v \},$$

obtained by adding the deficiency $\mathrm{def}_G v$ to the induced subgraph G(V-{v}), is precisely the undirected graph of the (n -1)-by-(n -1) coefficient matrix in the reduced system of n -1 equations. The graph G_v is called the v-elimination graph of G (Rose, Tarjan, & Lueker, 1976).

Now let P be any n-by-n permutation matrix and suppose we wish to eliminate for some $k < n$ the first through kth components of the vector Px in that order. Let α_1 through α_k denote the vertices in G, representing rows 1 through k of the matrix PMP^T, respectively. If the k-by-k leading block in the matrix PMP^T is nonsingular, then the fill-in created in the process of transforming the original system $Mx = b$ into the reduced system of n - k equations can be obtained by repeated application of Parter's method at vertices $\alpha_1, \alpha_2, \ldots, \alpha_k$, in that order.

To quantify this, let α denote the ordering

$$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k),$$

and let us set the original graph G to G_0 and introduce the following k elimination graphs:

$$G_i = (G_{i-1})_{\alpha_i}, \qquad i = 1, 2, \ldots, k.$$

The graph G_1 is the α_1 - elimination graph of G_0, G_2 is the α_2 - elimination graph of G_1, and so on up to G_k, which is the α_k - elimination graph of G_{k-1}. Then by repeated application of Parter's method it follows that each edge in the set of edges defined by

$$F_G\alpha = \mathrm{def}_{G_0}\alpha_1 \cup \mathrm{def}_{G_1}\alpha_2 \cup \ldots \cup \mathrm{def}_{G_{k-1}}\alpha_k ,$$

corresponds exactly to an element of the fill-in produced in the process of generating the reduced system of n - k equations (assuming no cancellation of nonzero elements). Using this notation, we are in position to state the sparsity-preserving property of an interior clique.

Theorem 2. Let G(U) be any clique with nonempty interior in G = (V, E), and let α be any ordering of the vertices in the interior set int(U). Then

$$F_G\alpha = \emptyset.$$

Proof. The proof is by induction on the size k of the ordering α. Since each vertex in an interior clique is simplicial, all k vertices in α are simplicial vertices in G. This means that the deficiency of vertex α_1 in G_0 is empty since α_1 is a simplicial vertex in G, and so the theorem holds for k = 1. Suppose the assertion holds for any ordering $\alpha' = (\alpha_1, \alpha_2, \ldots, \alpha_i)$ with $i < k$. Then $F_G\alpha' = \emptyset$ and so the elimination graph G_i is a subgraph of the original graph G. This means that α_{i+1} is a simplicial vertex in G_i since α_{i+1} is a simplicial vertex in G. Thus the deficiency of vertex α_{i+1} in G_i is empty and so the assertion holds for the ordering $(\alpha_1, \alpha_2, \ldots, \alpha_{i+1})$. By the induction hypothesis, the proof is now complete.

We will return to this result in subsequent developments when we deal with matrix interpretations of the graph-theoretic results derived in this work.

4. A METHOD FOR ISOLATING ALL INTERIOR CLIQUES IN A GRAPH

The main property of the concept of core of a clique is highlighted in the following result.

Theorem 3. Let $\Pi = (V_1, V_2, \ldots, V_k)$ be any clique partition in G = (V, E). Then for any clique G(U) in G, the following relation holds.

$$\text{int}(U) \subseteq \bigcup_{i=1}^{k} \text{cor}(V_i) .$$

Proof. If G(U) is a clique with empty interior, we have nothing to prove. Suppose G(U) is a clique with nonempty interior, and let C_Π denote the union of the k sets of vertices $\text{cor}(V_1)$ through $\text{cor}(V_k)$. Assume for contradiction that the assertion of the theorem does not hold. Then we get $\text{int}(U) \cap (V - C_\Pi) \neq \emptyset$, and so for some element V_i in the clique partition Π we have $\text{int}(U) \cap (V_i - \text{cor}(V_i)) \neq \emptyset$. Let u be any vertex in the intersection $\text{int}(U) \cap (V_i - \text{cor}(V_i))$ and let v be any vertex in $\text{cor}(V_i)$. Since v is a vertex in the clique $G(V_i)$, every vertex in the set $V_i - \{v\}$ is adjacent to v. This means that the vertex u in int(U) is adjacent to v since u is in the set V_i. So by statement (b) of Lemma 1 we get $V_i \subseteq U$ since any vertex adjacent to a vertex in int(U) must be in U. Now one of the following two cases must hold.

Case 1. $v \in \text{int}(U)$. Then we get $\deg_G u = \deg_G v$ since both vertices u and v are in int(U). But since u is in V_i - cor(V_i) and v is in cor(V_i) we have $\deg_G u > \deg_G v$ and a contradiction.

Case 2. $v \notin \text{int}(U)$. Then $v \in U$ - int(U) since v is a vertex in V_i and $V_i \subseteq U$. Thus we get $\deg_G u < \deg_G v$ since u is in int(U) and v is in U - int(U). But since vertex u is in V_i - cor(V_i) and v is in cor(V_i) we have $\deg_G u > \deg_G v$ and a contradiction.

This completes the proof.

Since each vertex and each edge of a graph G = (V, E) forms a clique, consider a special case where each element V_i of a clique partition Π in G corresponds to either a vertex in V or an edge in E. If each element of Π corresponds to a vertex in V, then the application of Theorem 3 to Π provides no worthwhile information since cor(V_i) = V_i, which means that $C_\Pi = V$. On the other hand, if any element V_i of Π corresponds to an edge (v,w) in E with $\deg_G v \neq \deg_G w$, then by Theorem 3 we can immediately conclude that the vertex u with the strictly larger degree in the set of vertices $V_i = \{v,w\}$ can never be part of any interior clique in G since u is not in cor(V_i). This interpretation of Theorem 3 using edges of a graph directly leads to a result that identifies a set of vertices S such that all interior cliques in G are connected components of the induced subgraph G(V-S). A concise statement of this result follows.

Corollary 3.1. Let S be a set of vertices in G = (V, E) defined by

$$S = \{v \in V | \exists\ (v, w) \in E \text{ with } \deg_G v > \deg_G w\}\ .$$

Then every interior clique in G is a connected component of G(V-S).

Proof. Let G(U) be any clique in G with nonempty interior. Then the intersection of the sets int(U) and S must be empty if the assertion of the corollary holds. Assume for contradiction that the intersection of the sets int(U) and S is nonempty, and let v be any vertex in the intersection. Since v is in S, there exists in E an edge (v, w) such that $\deg_G v > \deg_G w$. Let U be the vertex set consisting of the two vertices v and w. Then G(U) is a clique with cor(U) = {w} and U - cor(U) = {v} since $\deg_G w < \deg_G v$. Now let Π be any clique partition in the graph G = (V, E) such that U is an element of Π. Then by Theorem 3 we have int(U) $\subseteq C_\Pi$ and so v is in the set C_Π since v is in int(U). But this is a contradiction since v is not in cor(U). Thus, we get int(U) $\cap$ S = $\varnothing$, which means that the interior clique G(int(U)) is a subgraph of G(V-S).

Assume for contradiction that G(int(U)) is not a connected component of G(V-S). Then by statement (b) of Lemma 1 the intersection ext(U) $\cap$ (V-S) must be nonempty. Let v be any vertex in the intersection of the sets ext(U) and (V-S) and let u be any vertex in int(U). Then there exists in E an edge (u, v) with $\deg_G v > \deg_G u$ since u is in int(U) and v is in ext(U). Thus, the vertex v is in the set S which is a contradiction since it was assumed that v is a vertex in V-S. This completes the proof.

Corollary 3.1 motivates a four-step method for exploiting the underlying structure of an arbitrary undirected graph G = (V, E). These steps are as follows:

1. Compute the set of vertices S;
2. Compute connected components of induced subgraph G(V-S);
3. Classify clique connected components of G(V-S);
4. Compute independent cliques in nonclique connected components of G(V-S).

In what follows, we give detailed algorithms for the solution of Problems 1 through 4. Initially, all algorithms will be presented in the Algol-like language adopted by Aho, Hopcroft, and Ullman (1976). The clique connected components computed in Problem 2 together with the independent cliques computed in Problem 4 form the parallel regions produced by this parallelization tool. Computer implementation of the parallelization tool using the linear algebra package Matlab is given in Kevorkian (1993).

5. COMPUTING THE SET OF VERTICES S

We solve Problem 1 in linear time by visiting the vertices and edges of a graph G = (V, E) in the following manner. We select and visit a vertex v. Then for each vertex w adjacent to v we do the following. If w has been visited previously, we pick another vertex adjacent to v. If w has not been visited previously, we compare the degrees at vertices v and w. If the degrees are equal, we pick another vertex adjacent to v. If the degrees are unequal, then the vertex with the strictly larger degree is marked as a vertex that belongs to the set S. This process is continued until all vertices in V have been visited.

The following algorithm computes the vertex set S in running time proportional to the number of vertices in V plus number of edges in E. The input to the algorithm is an undirected graph G = (V, E) represented by adjacency lists ADJ(v) for all v in V. We use a Boolean array SN setting SN(v) = 1 if and only if v is an element of the set S. We assume all vertices are initially marked "new" and all degrees have been computed and stored on the single array DEG.

```
procedure search:
begin
    for all v in V do
        begin
            mark v "old" ;
            for each vertex w on ADJ(v) do
                if w is marked "new" then
                    if DEG(v) ≠ DEG(w) then
                        if DEG(v) < DEG(w) then
                            SN(w) ← 1
                        else
                            SN(v) ← 1
        end;
    comment S = {v|SN(v) = 1}
end
```

By construction, the set of vertices S computed in the graph G = (V, E) by procedure search will be empty if and only if G is a regular graph. Note that every vertex in a regular graph has minimum degree, and so computing independent cliques in G (last step of the parallelization tool) makes good sense for parallel computation. If the set of vertices S is nonempty and G(V-S) is a connected graph, then one can easily show that the set of vertices V-S satisfies the following equality

$$V - S = \{ u \mid \deg_G u = \min_{v \in V} \deg_G v \}.$$

This means that every vertex in the set V-S has minimum degree in G, and so computing independent cliques in G(V-S) is a sensible strategy for parallel computation.

One other connection between the set S and vertices with minimum degree is highlighted in the next result.

Lemma 2. For any graph G = (V, E), if a vertex v in V has minimum degree in G then v is a vertex in the set V-S.

Proof. If v is a vertex in V with minimum degree in G, then for any edge (v, w) incident with v we have $\deg_G v \leq \deg_G w$, which means that v can never be in the set S. This completes the proof.

6. COMPUTING CONNECTED COMPONENTS OF INDUCED SUBGRAPH G(V-S)

Problem 2 is solved in linear time using the depth-first search method (Tarjan, 1972; Aho et al., 1976). The connected components of the induced subgraph G(V-S) are placed consecutively on the single array QUEUE. Pointers to the starting vertices of the connected components are placed on the single array IQUEUE. Computation of the array IQUEUE is aided using two integers LEAF and ROOT where LEAF is the pointer to the last vertex placed on array QUEUE and ROOT is the pointer to the starting vertex of the most recently computed connected component. The entire algorithm together with the procedure component(v), called by dfs, is given below. We assume both arrays QUEUE and IQUEUE are initially set to empty and the integer LEAF is set to zero.

```
procedure dfs:
for all v in V do
    if SN(v) = 0 then
        if v is marked "new" then
            begin
                mark v "old" ;
                add v to end of QUEUE ;
                LEAF ← LEAF + 1;
                ROOT ← LEAF ;
                add ROOT to end of IQUEUE ;
                RANKE ← 0 ;
                NGU ← empty ;
                component(v) ;
                RANKU ← LEAF - ROOT + 1 ;
                RANKN ← |NGU| ;
                for all w on NGU do TEST(w) ← 0 ;
                classify
            end


procedure component(v):
for each vertex w on ADJ(v) do
    if SN(w) = 0 then
        begin
            RANKE ← RANKE + 1 ;
            if w is marked "new" then
                begin
                    mark w "old";
                    add w to end of QUEUE ;
                    LEAF ← LEAF + 1 ;
                    component(w)
                end
        end
    else
        if TEST(w) = 0 then
            begin
                add w to end of NGU ;
                TEST(w) ← 1
            end
```

Since ROOT is the pointer to the starting vertex v of connected component G(U) and LEAF is the pointer to the last vertex placed on array QUEUE, at the completion of the call to procedure component(v) we have

$$|U| = LEAF - ROOT+1.$$

We use the integer RANKU in dfs to compute |U|.

The integer RANKE (initialized in dfs and computed in the recursive procedure component(v)) keeps count of the edges encountered in a connected component computed by dfs. Therefore if G(U) denotes a connected component of G(V-S), then at the completion of G(U) in dfs we must have RANKE = 2|E(U)| since the depth-first search method encounters every edge of an undirected graph exactly twice (Aho et al., 1976). Therefore a connected component G(U) computed by dfs is a clique if and only if the following equality holds

$$RANKE = RANKU \times (RANKU - 1),$$

since a clique with |U| vertices will have $|U| \times (|U| - 1)/2$ edges. This equality constitutes the algebraic relation we use (in procedure classify) for categorizing the connected components of the induced subgraph G(V-S) as cliques graphs and non-cliques graphs.

While computing a connected component G(U) in procedure component(v), we also compute a set of vertices N_GU defined by

$$N_GU = \{w \in V - U | \text{ w is adjacent to a vertex in U}\} .$$

The set N_GU consists of all vertices in V-U that are adjacent to some vertex in U. We call the set of vertices N_GU the neighborhood of U in G. The single array NGU (initialized to empty in dfs and computed in component(v)) stores the vertices in the neighborhood N_GU one at a time. We use the Boolean array TEST in component(v) to ensure that no vertex in the set N_GU is added more than once to the array NGU. We assume the Boolean array TEST contains all 0's initially.

At the completion of the recursive procedure component(v) in dfs, we use the integer RANKN to compute the size of the neighborhood N_GU and we reset the Boolean array TEST so that all the 1's in TEST are set back to zero. Subsequently we call procedure classify to execute step 3 of our method.

7. CLASSIFYING CLIQUE CONNECTED COMPONENTS OF G(V-S)

This section classifies the cliques in a graph G = (V, E) into four distinct types with the interior cliques forming one of these four types. The result leading to this classification of cliques is given next.

Theorem 4. For any $U \subset V$ in G = (V, E), G(U) is an interior clique if and only if the following three conditions are satisfied:

(a) for any $u \in U$, $adj_G u = N_G U \cup (U - \{u\})$,
(b) for any $v \in N_G U$, v is adjacent to all other vertices in $N_G U$,
(c) for any $u \in U$ and any $v \in N_G U$, $deg_G u < deg_G v$.

Proof. Suppose conditions (a), (b), and (c) hold. By condition (a), each vertex in U is adjacent to all vertices in $N_G U$ and all other vertices in U. Thus by the combination of conditions (a) and (b), each vertex in the set of vertices C defined by $C = U \cup N_G U$ is adjacent to all other vertices in C. This means that the induced subgraph G(C) is a clique. Furthermore, by condition (a) we have $adj_G u = C - \{u\}$, for all u in U, which means that

$$deg_G u = |C| - 1, \qquad \text{for any } u \in U,$$

and so we obtain $U \subseteq int(C)$. But by condition (c), no vertex in $N_G U$ can be in the set int(C), and so we have U = int(C), which means that G(U) is an interior clique.

Suppose G(U) is an interior clique. Then there is in G a clique G(C) such that U = int(C). This means that $adj_G u = C - \{u\}$ for any vertex u in U, and so we get $N_G U = C - U$. Subsequently, by combining these two equalities, we obtain the statement in condition (a). Also, since G(C) is a clique and $N_G U \subset C$, we obtain the statement in condition (b). Finally, since $N_G U = C - U$ and $deg_G u < deg_G v$ for any vertex u in U and any vertex v in C-U, we have the statement in condition (c). This completes the proof of the theorem.

As an immediate consequence of Theorem 4, we obtain the following series of results pertaining to the set of vertices S defined in Corollary 3.1.

Corollary 4.1. Let G(U) be any connected component of G(V-S). Then G(U) is an interior clique if and only if conditions (a) and (b) in Theorem 4 hold.

Proof. Suppose conditions (a) and (b) in Theorem 4 hold. Assume for contradiction that condition (c) in Theorem 4 does not hold. Then for some vertex u in U and some vertex v in $N_G U$ we have $deg_G u \geq deg_G v$. But by conditions (a) and (b) we

have $U \subseteq \text{int}(C)$. This means that $u \in \text{int}(U)$ and so we obtain $v \in \text{int}(C)$ since $\deg_G v \leq \deg_G u$. However, as G(U) is a connected component of G(V-S) we have $N_G U \subseteq S$, which means that vertex v is also in S. But this is a contradiction since by Corollary 3.1 no vertex in V can be in both S and int(U). Therefore, condition (c) holds and so by Theorem 4 the connected component G(U) is an interior clique.

Suppose G(U) is an interior clique. Then by Theorem 4 both conditions (a) and (b) hold. This completes the proof of the corollary.

Corollary 4.2. Let G(U) be any connected component of G(V-S). Then G(U) is an interior clique if the following two conditions hold.

(a) G(U) is a clique,
(b) $|N_G U| = 1$.

Proof. Suppose conditions (a) and (b) hold. Since $N_G U$ is nonempty, there is a vertex u in U and a vertex v in $N_G U$ such that u is adjacent to v. By condition (a) vertex u is adjacent to all other vertices in U. By condition (b) vertex u is not adjacent to any vertex in V - U - $N_G U$ and so we get $\deg_G u = |U|$. Assume now for contradiction that there is a vertex u' in U such that u' is not adjacent to vertex v. Then u' is not adjacent to any vertex in V - U - $N_G U$ since NGU = {v}, and so by condition (a) we get $\deg_G u' = |U| - 1$. Also, by condition (a) vertex u is adjacent to u' and so (u, u') is an edge in E with $\deg_G u' < \deg_G u$, which means that u is a vertex in the set of vertices S. However, this is a contradiction since u is a vertex in U and G(U) is a connected component of the induced subgraph.G(V-S). Therefore, each vertex in U has degree |U| and so $\text{adj}_G u = \{v\} \cup U - \{u\} = N_G U \cup (U - \{u\})$ for all u in U. Therefore condition (a) in Theorem (4) holds. Also, condition (b) in Theorem 4 holds since $N_G U$ consists of a single vertex. Consequently, by Corollary 4.1 the proof is complete.

Corollary 4.1 classifies each clique G(U) in a graph G into one of four distinct types. These are as follows: Type C_1 - G(U) satisfies both conditions (a) and (b); Type C_2 - G(U) satisfies condition (a) and not (b); Type C_3 - G(U) satisfies condition (b) and not (a); and Type C_4 - G(U) satisfies neither condition (a) nor (b). Figure 3 gives the roadmap of connections between these four types of cliques. We call the cliques of types C_1 and C_2 semi-interior cliques, whereas cliques of type C_2 are called strictly semi-interior. By these definitions, all cliques in a graph are either semi-interior or nonsemi-interior. Moreover, If any clique G(U) is semi-interior, then G(U) is either interior or strictly semi-interior.

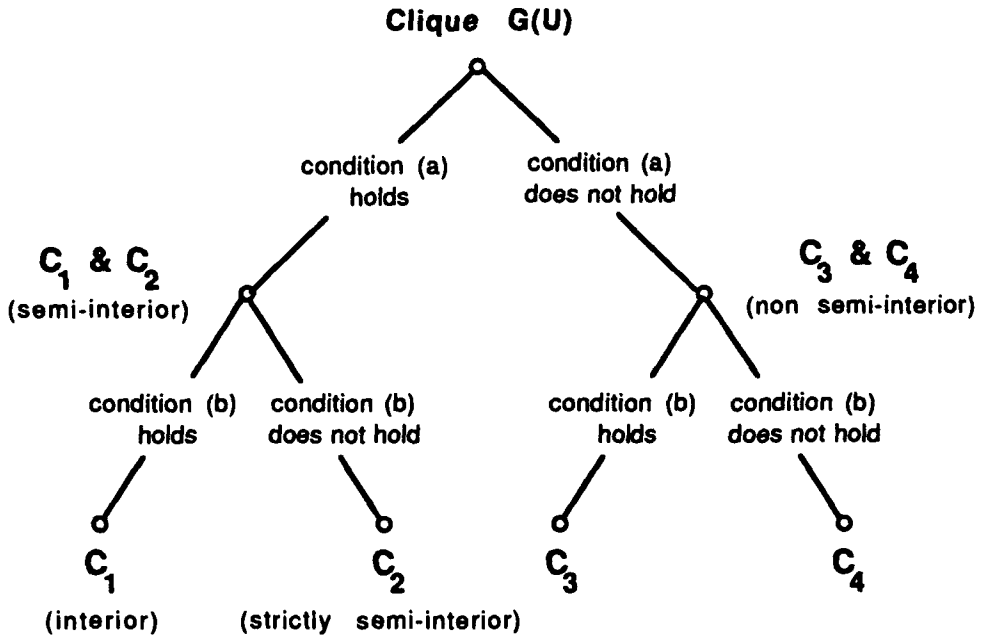

Figure 3. The roadmap linking the four distinct types of cliques in a graph.

Our next result gives a simple necessary and sufficient condition for the existence of a semi-interior clique. This condition forms the key algebraic relation for finding all semi-interior cliques in the induced subgraph G(V-S).

Corollary 4.3. For any $U \subset V$ in $G = (V, E)$, G(U) is a semi-interior clique if and only if the following condition holds.

$$\deg_G u = |N_G U| + |U| - 1, \qquad \text{for all } u \in U.$$

Proof. Suppose G(U) is a semi-interior clique. Then condition (a) in Theorem 4 holds and thus the assertion holds.

Assume now the assertion in the corollary holds. Then for any u in U, we get $|adj_G u| = |N_G U| + |U| - 1$, and so we have $|adj_G u| = |N_G U| + |U - \{u\}|$. But by the definition of $N_G U$, the two sets of vertices $N_G U$ and U are disjunct Therefore we obtain $|adj_G u| = |N_G U \cup (U - \{u\})|$, and so condition (a) in Theorem 4 holds. This completes the proof.

The algebraic relation in Corollary 4.3 is obtained in the process of computing the connected components of the induced subgraph G(V-S), and so the entire family of semi-interior cliques in the induced subgraph G(V-S) is readily computed by applying the equality in Corollary 4.3 to each vertex u in a clique connected component of G(V-S).

The categorization of the connected components of G(V-S) into cliques and noncliques and the computation of the family of semi-interior cliques in G(V-S) are carried out in the algorithm presented below.

```
procedure classify:
if RANKE = RANKU × (RANKU-1) then
    if RANKN = 1 then
        add 1 to end of TYPE
    else
        for each u on QUEUE(ROOT:LEAF) do
            if DEG(u) ≠ RANKN+RANKU-1 then
                add 3 to end of TYPE
            else
                add -2 to end of TYPE
else
    begin
        add 0 to end of TYPE ;
        for each u on QUEUE(ROOT:LEAF) do
            mark u "new" ;
        cliques
    end
```

At the invocation of classify, ROOT is the pointer to the starting vertex of the connected component G(U) on QUEUE, and LEAF is the pointer to the last vertex and so QUEUE(ROOT:LEAF) forms the part of array QUEUE containing the set of vertices U. The single array TYPE provides all information on whether a connected component G(U) of G(V-S) is a clique or not, and on what type of clique G(U) is. If G(U) denotes the kth connected component computed by dfs, then we use the following convention in procedure classify to categorize and classify connected components.

$$\text{TYPE(k)} = \begin{cases} 0 & G(U) \text{ is not a clique} \\ 1 & G(U) \text{ is an interior clique (type } C_1) \\ 2 & G(U) \text{ is a strictly semi-interior clique (type } C_2) \\ -2 & G(U) \text{ is a semi-interior clique (type } C_1 \text{ or } C_2) \\ 3 & G(U) \text{ is a clique of either type } C_3 \text{ or type } C_4 \end{cases}$$

The initial task in procedure classify is to categorize the connected components of G(V-S) into cliques and noncliques. For any connected component G(U) of G(V-S), if RANKE ≠ RANKU × (RANKU -1), then G(U) is not a clique. Subsequently, we add 0 to the end of array TYPE, mark all vertices in U "new," and then call procedure cliques to compute independent cliques in G(U). Since our immediate interest is to classify the clique components of G(V-S), we defer the description of procedure cliques to later on.

Assume the equality RANKE = RANKU × (RANKU -1) holds. Then the connected component G(U) is a clique. At this stage, procedure classify proceeds with the application of Corollaries 4.2 and 4.3 to G(U).

If |NGU| = 1, then both conditions in Corollary 4.2 are satisfied, which means that G(U) is an interior clique. Subsequently, we add 1 to the end of array TYPE and return to the calling procedure dfs to compute the next connected component.

Suppose |NGU| > 1. Then condition (a) in Corollary 4.2 does not hold. We then proceed with the application of Corollary 4.3 to G(U). If the equality in Corollary 4.3 does not hold, then G(U) is not a semi-interior clique, which means that G(U) is neither an interior clique nor a strictly semi-interior clique. Subsequently, we add 3 to the end of array TYPE and return to the calling procedure dfs. Assume the equality in Corollary 4.3 holds. Then G(U) is a semi-interior clique. Subsequently we add -2 to the end of array TYPE and return to the calling procedure dfs.

At the completion of procedure dfs, the computation of all semi-interior cliques in the induced subgraph G(V-S) is complete.

7.1. MATRIX INTERPRETATION OF THE CLIQUE CLASSIFICATION

Let M be any square symmetric matrix, and let P be any permutation matrix such that PMP^T is a 2-by-2 block matrix

$$PMP^T = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix} \tag{5}$$

in which the leading block A (called henceforth the pivot block) is square and non-singular. Assume now that there exists an upper triangular matrix U with nonzero diagonal entries such that

$$A = U^T U . \tag{6}$$

Then the block matrix PMP^T can be written in the block product form

$$PMP^T = \begin{bmatrix} U^T & 0 \\ X^T & I \end{bmatrix} \begin{bmatrix} U & X \\ 0 & D - X^T X \end{bmatrix} \tag{7}$$

in which I is an identity matrix, the 0's are zero matrices, and the block X is the solution of the following triangular system with multiple right-hand sides

$$U^T X = B . \tag{8}$$

The matrix $D - X^T X$ is the familiar Schur complement of A in PMP^T. The correctness of the block product form (7) can be verified by multiplication.

In the majority of sparse matrix problems, the blocks B and D in the 2-by-2 block matrix PMP^T are sparse matrices whereas their counterparts X and $D - X^T X$ in the product form of PMP^T can be extremely dense if the permutaion matrix P is not chosen carefully. The sparser the block X and the Schur complement $D - X^T X$ are, the more efficient is the computation process.

Given any block diagonal pivot block A with full diagonal blocks, we show that the sparsity of block X and the Schur complement $D - X^TX$ are fully dictated by the clique classification presented earlier. But first we need new notation.

For any square or nonsquare matrix H, the structure of H is a 0 - 1 matrix str(H) obtained by replacing each nonzero entry of H with '1.'

With this notation, it is easy to see that the block X and the Schur complement $D - X^TX$ have the same sparsity as the blocks B and D if and only if str(X) = str(B) and $str(D - X^TX) = str(D)$, respectively.

Suppose G = (V, E) is the graph of the symmetric matrix M, and let C be the set of vertices in V representing the rows of the pivot block A in PMP^T. Then for the case where the pivot block A is a full matrix (or when G(C) is a clique), we have the following result pertaining to the structures of block X and the Schur complement $D - X^TX$. We assume no cancellation of nonzero elements takes place in equation (7).

Theorem 5. Suppose G(C) is a clique. Then the following statements are true.

(a) If G(C) is type C_1 clique (interior), then

$$str(X) = str(B),$$
$$str(D - X^TX) = str(D).$$

(b) If G(C) is type C_2 clique (strictly semi-interior), then

$$str(X) = str(B),$$
$$str(D - X^TX) \neq str(D).$$

(c) If G(C) is type C_3 clique, then

$$str(X) \neq str(B),$$
$$str(D - X^TX) = str(D).$$

(d) If G(C) is type C_4 clique, then

$$str(X) \neq str(B),$$
$$str(D - X^TX) \neq str(D).$$

Proof. Suppose the clique G(C) is semi-interior. Then G(C) is either a type C_1 clique or type C_2. Assume for contradiction that $str(X) \neq str(B)$. Then for some row i and some column j of blocks X and B we have $x_{ij} \neq 0$ and $b_{ij} = 0$. Thus, there exists a vertex u in C and a vertex v in $N_G C$ such that the pair (u, v) is an edge in the complement G^* of G. This means that the pair (u, v) is not an edge in E. But this is a contradiction since each vertex of any semi-interior clique G(U) is adjacent to every vertex in the neighborhood of U. Therefore, the first equality in both statements (a) and (b) holds. As a direct consequence, the first equality in statements (c) and (d) will also hold since type C_3 and type C_4 cliques are nonsemi-interior cliques.

Now suppose G(C) is any clique such that $G(N_G C)$ is a clique. Then the clique G(C) is either type C_1 or type C_3. Assume for contradiction that $str(D - X^T X) \neq str(D)$. Then there exist two distinct vertices v and w in $N_G C$ such that the pair (v, w) is an edge of the complement G^* of G, which means that (v, w) is not an edge in E. However, this is a contradiction since $G(N_G C)$ is a clique. Thus the second equality in both statements (a) and (c) holds. As a direct consequence, the second equality in statements (b) and (d) will also hold since for any type C_2 and any type C_4 clique G(U) the induced subgraph $G(N_G U)$ is not a clique. This completes the proof of statements (a) through (d).

Figure 4 gives a graphical illustration of Theorem 5 by showing the block X and the Schur complement $D - X^T X$ in dark whenever $str(X) \neq str(B)$ and $str(D - X^T X) \neq str(D)$, respectively.

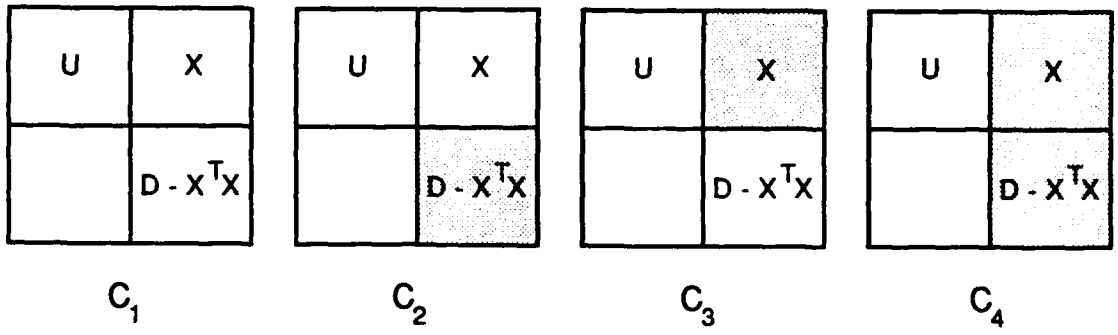

Figure 4. A graphical illustration of Theorem 5.

8. COMPUTING INDEPENDENT CLIQUES IN A NONCLIQUE CONNECTED COMPONENT

Given any nonclique connected component G(U) of the induced subgraph G(V-S), procedure cliques computes a vertex partition

$$\Pi' = (U_1, U_2, \ldots, U_t, S'), \tag{9}$$

satisfying the following two conditions:

(1) $G(U_i)$ is a maximal clique in subgraph of G induced by the set of vertices

$$Y_i = U - \bigcup_{k=1}^{i-1} (U_k \cup N_{G(U)} U_k), \qquad i = 1, 2, \ldots, t,$$

(2) each u in S' is a vertex in the neighborhood of some U_i in the partition.

By condition (1), $G(U_1)$ is a maximal clique in $G(Y_1)$ (where $Y_1 = U$), $G(U_2)$ is a maximal clique in $G(Y_2)$ and such that no vertex in U_2 is adjacent to a vertex in U_1, $G(U_3)$ is a maximal clique in $G(Y_3)$ and such that no vertex in U_3 is adjacent to a vertex in both U_1 and U_2, and so forth. This means that the t cliques computed in procedure cliques are independent cliques with maximality properties as defined in condition (1). By condition (2), no part of the connected component G(U) contains a clique that is independent of the cliques $G(U_1)$ through $G(U_t)$, and so procedure cliques computes as many independent cliques as possible.

The independent cliques computed in procedure cliques are initially placed one at a time on a single array CLQS (shortening for cliques) while the set of vertices S' is placed on another single array NBRS (shortening for neighborhoods). At the completion of the vertex partition Π', the part of array QUEUE containing the set of vertices U is replaced by the array [CLQS, NBRS].

To begin with, procedure cliques uses the starting vertex of the connected component G(U) as the starting vertex of the first independent clique that gets computed in cliques. This way the pointer ROOT to the starting vertex of G(U) becomes the pointer to the starting vertex of the first independent clique placed on array QUEUE. The pointers to the starting vertices of the remaining t elements of the vertex partition are determined while computing the partition and are placed on the array IQUEUE. The entire algorithm is as follows.

```
procedure cliques:
begin
      CLQS ← empty ;
      NBRS ← empty ;
      CLQROOT ← ROOT ;
      TAIL ← 0 ;
      for each v on QUEUE(ROOT:LEAF) do
          if v is marked "new" then
              begin
                    mark v "old" ;
                    ADJCNT ← empty ;
                    maxclq
              end;
      QUEUE(ROOT:LEAF) ← [CLQS, NBRS]
end
```

The integer CLQROOT determines the pointers to the starting vertices of the t+1 elements of the partition. Initially, CLQROOT is set to ROOT since the starting vertex of the connected component G(U) is also the starting vertex of the first independent clique placed on array QUEUE. The integer TAIL is used as the pointer to the most recent vertex placed on array CLQS. Since CLQS is initially empty, the integer TAIL is set to zero.

The initial task of procedure cliques is to select the starting vertices of the t independent cliques. Initially, all vertices on QUEUE(ROOT:LEAF) are marked "new." The first vertex v selected in the **for** loop is QUEUE(ROOT), which is the starting vertex of the connected component G(U). In general, suppose v is the most recently visited vertex in cliques. If v is marked "old," then vertex v was visited in cliques previously and so cliques picks the next vertex on QUEUE(ROOT:LEAF). Suppose v is marked "new." Then procedure cliques marks vertex v "old" (v is thus a 'visited' vertex in cliques); initializes the single array ADJCNT to empty and lastly it calls procedure maxclq.

The main objective of procedure maxclq is to compute an independent clique with starting vertex v. If v is adjacent to some vertex w in a previously computed independent clique (w is on array CLQS), then v is rejected in maxclq as a starting vertex. Subsequently, maxclq returns to procedure cliques to select another starting vertex. Otherwise the vertex v, selected in cliques, becomes the starting vertex of the next independent clique computed in procedure cliques. To distinguish between vertices placed on the two arrays CLQS and NBRS, procedure maxclq uses the Boolean array SN setting SN(u) = 1 if and only if a vertex u in U is placed on array NBRS. The entire procedure maxclq is as follows.

```
procedure maxclq:
begin
    for each vertex w on ADJ(v) do
        if SN(w) = 0 then
            if w is marked "new" then
                add w to end of ADJCNT
            else
                begin
                    comment v is rejected as starting vertex ;
                    add v to end of  NBRS ;
                    SN(v) ← 1 ;
                    return
                end;
    comment v is accepted as starting vertex ;
    add v to end of CLQS ;
    TEST(v) ← 1 ;
    RANKC ← 1 ;
    for each vertex u on ADJCNT do
        begin
            COUNT ← 0 ;
            mark u "old" ;
            for each vertex w on ADJ(u) do
                if TEST(w) = 1 then
                    COUNT ← COUNT+1;
            if COUNT = RANKC then
                begin
                    add u to end of CLQS ;
                    TEST(u) ← 1;
                    RANKC ← RANKC+1
                end
            else
                begin
                    add u to end of NBRS ;
                    SN(u) ← 1
                end
        end;
    HEAD ← TAIL+1 ;
    TAIL ← TAIL+RANKC ;
    for each u on CLQS(HEAD:TAIL) do TEST(u) ← 0 ;
    CLQROOT ← CLQROOT+RANKC ;
    add -CLQROOT to end of IQUEUE
end
```

The acceptance or rejection of vertex v as a starting vertex is accomplished in the top **for** loop in maxclq. If v is rejected as a starting vertex, maxclq returns to cliques to select the next starting vertex. If v is accepted, then the computed array ADJCNT contains every vertex in the independent clique that has v as its starting vertex. To show this, let w be any vertex on ADJ(v). Then one of the following two cases must hold.

Case 1. SN(w) = 1. Then w is either in S or on the array NBRS. If w is in S then w is not in U. If w is on NBRS, then w is not a vertex in a previously computed independent clique and so v is not rejected as a starting vertex. Therefore if SN(w) = 1, maxclq continues with the processing of the top **for** loop by visiting the next vertex on ADJ(v).

Case 2. SN(w) = 0. Then w is in the set U. If w is marked "new," maxclq places w on the array ADJCNT and continues with the processing of the loop by visiting the next vertex on ADJ(v). Suppose w is marked "old." Then vertex w has already been visited in procedure cliques and so w is either on array CLQS or on array NBRS. But since SN(w) = 0, vertex w is not on array NBRS and so w has to be on array CLQS. This means that v is adjacent to a vertex in a previously computed independent clique and so v is a vertex in the neighborhood of a previously computed independent clique. Consequently, maxclq rejects v as a starting vertex by placing v on array NBRS; setting SN(v) to 1 and then returning to procedure cliques to select the next starting vertex v on array QUEUE(ROOT:LEAF).

To distinguish between previously computed independent cliques and the currently computed independent clique, maxclq uses the Boolean array TEST setting TEST(u) = 1 if and only if vertex u is in the currently computed independent clique. The size of the most recently computed independent clique is determined using the integer RANKC. Thus when vertex v is added to CLQS at the completion of the top **for** loop (that is when v is accepted as starting vertex), TEST(v) is set to 1 and also RANKC is set to 1 since v is the very first vertex of the currently computed independent clique placed on array CLQS.

The computation of the independent clique with starting vertex v is completed with the aid of the second **for** loop in maxclq. Let C denote the set of vertices added to array CLQS at the most recent call to maxclq. Then at the completion of each pass of the second **for** loop we have COUNT = $| C \cap ADJ(u) |$ since $C = \{ x \in V \mid TEST(x) = 1 \}$. But by the construction of the integer RANKC we have RANKC = $|C|$ and so if the condition COUNT = RANKC is satisfied in the loop, then $C \subseteq ADJ(u)$, which means that vertex u is adjacent to each vertex in C, or equivalently, u is a vertex in the indpendent clique with starting vertex v. Hence if the condition COUNT = RANKC is satisfied, maxclq adds u to the array CLQS; sets TEST(u) to 1 and RANKC to RANKC+1 and then it continues with the processing of the loop. Suppose COUNT $\neq$ RANKC. Then vertex u is not adjacent to every vertex in C and so u is not a vertex in the independent clique that has v as starting vertex. However, u is adjacent to the starting vertex v, which means that u is a vertex in the neighborhood of the indepedent clique with starting vertex v. Consequently, maxclq adds u to the array NBRS; sets SN(u) = 1 and then continues with the processing of the **for** loop.

At the completion of the second **for** loop, the induced subgraph G(C) denotes the most recently computed independent clique. However, at the next call to maxclq, G(C) is a previously computed independent clique and so we need to reset the Boolean array TEST since each vertex u in a previously computed independent clique has TEST(u) = 0. Accordingly, procedure maxclq computes the integers HEAD and TAIL (pointers to the starting vertex and last vertex of G(C) on array CLQS, respectively), and then it uses them to reset the array TEST.

At the start of procedure maxclq, CLQROOT is the pointer to the starting vertex v of the independent clique G(C) computed at the finish of maxclq, and so the pointer to the starting vertex of the next independent clique is CLQROOT+RANKC since RANKC = |C|. The computation of the pointer CLQROOT to the next starting vertex is carried out in the one before the last statement in maxclq. It is worth noting that if G(C) is the last independent clique in G(U), then the integer CLQROOT computed in maxclq is the pointer to the starting vertex of the vertex set S' on array QUEUE. The last statement in maxclq adds the pointer CLQROOT (with a '-' appended to it) to the array IQUEUE. The '-' part of CLQROOT is used for distinguishing pointers added to IQUEUE in maxclq from pointers added to IQUEUE in procedure dfs. At the completion of maxclq, the program returns to procedure cliques to compute the next independent clique in G(U).

At the completion of the **for** loop in cliques, the computation of the partition Π' is complete. The array CLQS contains the sets of vertices U_1 through U_t in that order, while the other array NBRS contains the set of vertices S'. At this point, procedure cliques replaces the part QUEUE(ROOT:LEAF) of array QUEUE by the structured array [CLQS, NBRS] and returns to procedure dfs to compute the next connected component of the induced subgraph G(V-S).

9. COMPLEXITY OF PARALLELIZATION TOOL ROADMAP

Procedures search, dfs, classify, and cliques dictate the computational complexity of our parallelization tool. We will analyze the complexities of these four procedures individually to prove the linear-time complexity of the parallelization tool.

Procedure search visits each vertex in V exactly once, and each edge in E at most twice. Thus the total time spent in search is proportional to the number of vertices in V plus the number of edges in E.

If we exclude the call to classify in procedure dfs, then dfs is an implementation of the depth-first search method (Tarjan, 1972) for computing all connected components of the induced subgraph G(V-S). Thus the total time spent in dfs, excluding calls to classify, is proportional to the number of vertices in V plus the number of edges in E.

For each connected component G(U) computed in roadmap, procedure dfs calls classify. At the invocation of classify, the integer ROOT is the pointer to the starting vertex of G(U) and LEAF is the pointer to the end vertex. One of the following two cases must hold:

(a) G(U) is a clique. Then procedure classify applies both Corollaries 4.2 and 4.3 to G(U). The time required to apply Corollaries 4.2 and 4.3 is a constant plus time proportional to the size of the array U. Thus the time spent in classify is a constant plus time proportional to the number of vertices on U.

(b) G(U) is not a clique. Then classify calls procedure cliques. For each vertex v in U marked "new," cliques marks v "old," initializes array ADJCNT to empty and then calls procedure maxclq. In the top **for** loop of maxclq, every edge incident with v is visited at most once. If v is rejected as a starting vertex, maxclq returns to cliques to pick another vertex, which is marked "new." If v is accepted as a starting vertex, then maxclq computes an independent clique $G(U_i)$ with starting vertex v. The computation of the set $U_i - \{v\}$ is accomplished by visiting each vertex on the array ADJCNT exactly once. For each u on ADJCNT, maxclq visits every edge incident with u exactly once. At the completion of the independent clique $G(U_i)$, maxclq visits every vertex of $G(U_i)$ once more in order to update the Boolean array TEST. Subsequently, maxclq returns to cliques to pick the starting vertex of the next independent clique. Note that all vertices placed on array ADJCNT are marked "old" at the completion of the call to maxclq, and so no vertex in V has its edges visited more than once. Also, no vertex in V is placed on the arrays ADJCNT and NBRS more than once since each vertex placed on ADJCNT must b e marked "new," and each vertex placed on the array NBRS must have separator number equal to 1. Therefore, the total time spent in maxclq is a constant plus time proportional to the number of vertices in U plus the number of edges in E(U). At the completion of the **for** loop in cliques, each vertex in U is visited once more to reorder U on array QUEUE. Therefore, the total time spent in cliques including all calls to maxclq is a constant plus time proportional to the number of vertices in U plus the number of edges in E(U).

By construction, the sets of vertices and the sets of edges in the connected components of G(V-S) form partitions of the vertex set V-S and the edge set E(V-S), respectively. Thus the total time spent in dfs including all calls to classify is a constant plus time proportional to the number of vertices in V plus the number of edges in E, and thus the parallelization tool has linear-time complexity. This completes the complexity analysis.

10. MATRIX INTERPRETATION OF VERTEX PARTITION

For any graph G = (V, E), let S be the set of vertices computed in procedure search and let $G(V_1)$ through $G(V_k)$ denote the connected components of G(V-S) computed in procedure dfs. By construction, the (k+1) tuple Π defined by

$$\Pi = (V_1, V_2, \ldots, V_k, S) \tag{10}$$

is a vertex partition in G. However, this vertex partition is distinctly different from an arbitrary vertex partition in G since no vertex in any element V_i of the partition Π is adjacent to a vertex in any other element V_j of the partition. By this property of the vertex partition Π it follows that the graph G_Π with respect to Π takes the star-shaped form shown in figure 5 with S as the root vertex and V_1 through V_k as the leaf vertices.

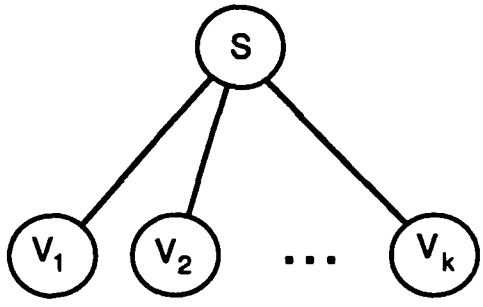

Figure 5. The graph G_Π with respect to partition Π.

Suppose G is the undirected graph of the n-by-n symmetric matrix M, and let A_{ii} be a square principal submatrix in M such that its rows correspond to the vertices in the ith element V_i of the vertex partition Π. Then by the structure of the star-shaped graph G_Π, there exists a permutation matrix P such that PMP^T has the following (k+1)-by-(k+1) block form

$$PMP^T = \begin{bmatrix} A_{11} & & & B_1 \\ & A_{22} & & B_2 \\ & & \ddots & \vdots \\ & & A_{kk} & B_k \\ B_1^T & B_2^T & \cdots & B_k^T \quad D \end{bmatrix} \qquad (11)$$

Since the set of vertices V-S consists of the union of the sets V_1 through V_k, the rows of the diagonal block D in PMP^T correspond to the vertices in the set S.

The origins of block bordered diagonal matrices can be traced to early works of Gabriel Kron (1958). In recent years, the use of block bordered diagonal forms in scientific and engineering computing has become an area of active research (Zhang et al., 1992). A main motivation has been the exploitation of parallelism using parallel architecture computers. The methodologies used for computing block bordered diagonal forms comes under a variety of names depending on the particular discipline. Examples include clustering (power system network problems [Ogbuobiri et al., 1970] and electrical circuits analysis and synthesis [Branin, 1975; Chua & Chen, 1976]), substructuring (structural engineering [Noor, Kamel, and Fulton, 1978]), macromodeling (VLSI circuit design [Rabbat, Sangiovanni-Vincentelli & Hsieh, 1979]) and domain decomposition [Chan et al., 1989; Glowinski et al., 1992] (solution of partial differential equations). In most of these methodologies, matrices are cast into block bordered diagonal forms by relying on the expertise of the modelers in the given discipline. In very special cases such as when the underlying graph of the problem is planar, block bordered diagonal forms can be obtained using nested dissection (George, Poole, and Voigt, 1978). The method developed in this work produces block bordered

diagonal forms for all types of graphs, and without any reliance on human expertise.

Without any loss of generality, assume the connected components $G(V_1)$ through $G(V_p)$ are cliques and the remaining components $G(V_{p+1})$ through $G(V_k)$ are not cliques. Then the leaf vertices of the graph G_Π can be grouped into two disjunct parts as shown in figure 6.

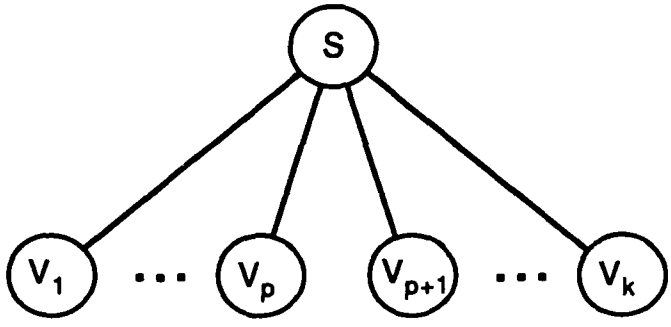

Figure 6. Categorization of leaf vertices of graph G_Π into cliques and noncliques.

The key objective of the procedure cliques is to explore the underlying structure of each nonclique connected component of G(V-S). Through this application of procedure cliques, the following two changes occur in the graph G_Π. First, the number of leaf vertices (parallel regions) will generally increase. Second, the graph corresonding to each leaf vertex of G_Π is a clique. This way we obtain another star-shaped graph of the form in figure 5, where each leaf vertex corresponds to a clique in G.

To illustrate this part of the work, let G(U) denote the nonclique connected component $G(V_{p+1})$ of the induced subgraph G(V-S). Then by replacing the (p+1) th element of the vertex partition Π in equation (10) by the vertex partition Π' in equation (9), we obtain another vertex partition Π'' defined by

$$\Pi'' = (V_1, \ldots, V_p, U_1, \ldots, U_t, S', V_{p+2}, \ldots, V_k, S) .$$

The graph $G_{\Pi''}$ with respect to this new vertex parttition Π'' is shown in figure 7.

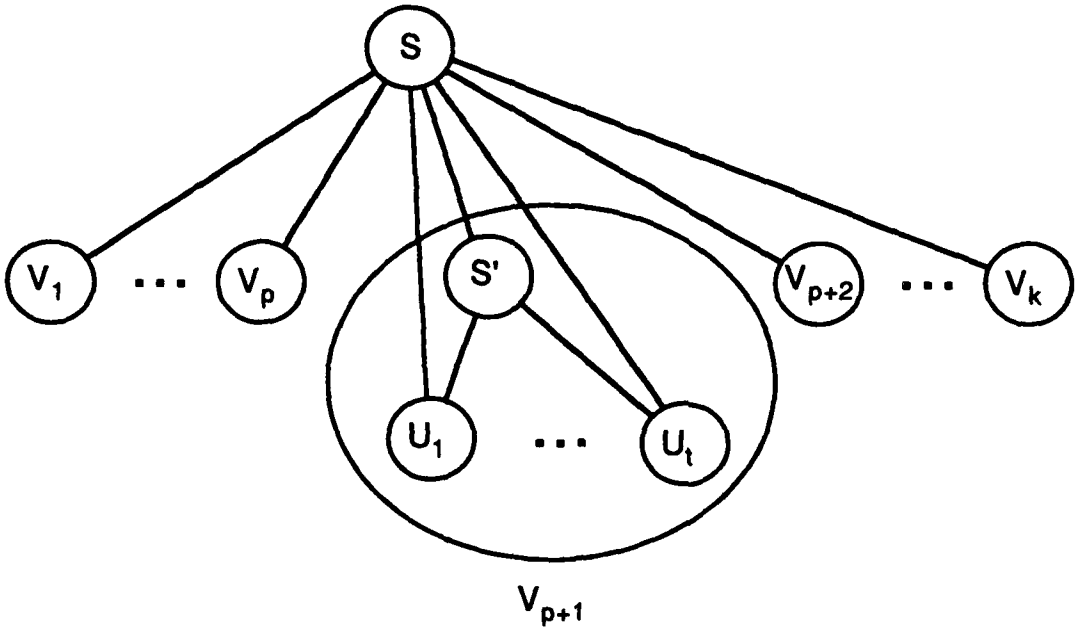

Figure 7. The graph $G_{\Pi''}$ with respect to the partition Π''.

By the structure of the star-shaped graph $G_{\Pi''}$, it is evident that if the elements S and S' of the partition Π'' are combined together to form the following vertex partition

$$\Pi''' = (V_1, \ldots, V_p, U_1, \ldots, U_t, V_{p+2}, \ldots, V_k, S \cup S'),$$

then the graph $G_{\Pi'''}$ with respect to the new vertex partition Π''' takes the star-shaped form shown in figure 8 with SUS' as the root vertex and $V_1, \ldots, V_p, U_1, \ldots, U_t, V_{p+2}, \ldots, V_k$ as the leaf vertices.

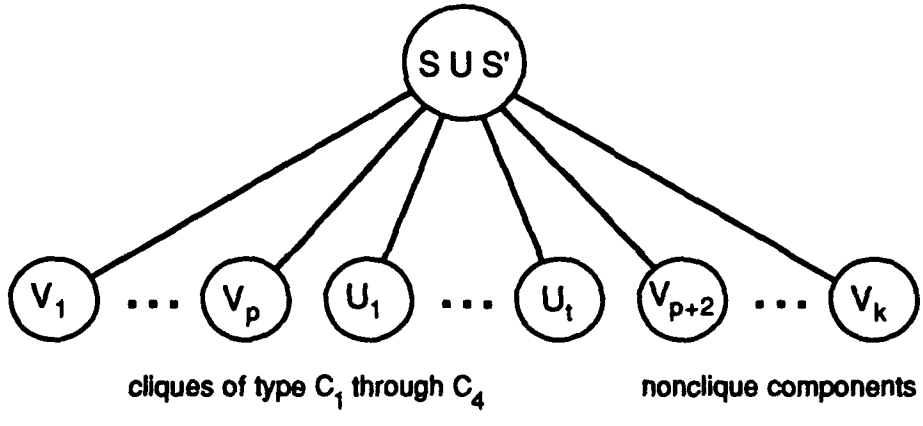

Figure 8. The graph $G_{\Pi'''}$ with respect to the partition Π'''.

Comparison of the graphs G_Π and $G_{\Pi'''}$ provides a number of worthwhile facts. To begin with, both G_Π and $G_{\Pi'''}$ are star-shaped graphs. The graph $G_{\Pi'''}$, however, has additional t-1 leaf vertices and such that each of these leaf vertices corresponds to a clique in G. Furthermore, the formation of the graph $G_{\Pi'''}$ eliminates a leaf vertex corresponding to a nonclique connected component of G(V-S), and so by applying the procedure cliques to the remaining p -1 nonclique connected components of G(V-S), we must obtain a partition Π^* such that G_{Π^*} is a star-shaped graph in which every leaf vertex corresponds to a clique in G.

To derive the partition Π^*, let S_i denote the set of vertices placed on array NBRS at the end of the application of procedure cliques to the nonclique connected component $G(V_i)$ and let S^* be the set of vertices defined by

$$S^* = S \cup \left(\bigcup_{i=p+1}^{k} S_i \right).$$

Then we have the following result.

Theorem 6. For any graph G = (V, E), the set of vertices S^* satisfies the following two conditions.

(a) Every connected component of $G(V-S^*)$ is a clique.
(b) Every interior clique in G is a connected component of $G(V - S^*)$.

Proof. Condition (a): Let $G(V_i)$ be any nonclique connected component of G(V-S). Then by the construction of the vertex set S_i, every connected component of the induced subgraph $G(V_i - S_i)$ is a clique, for i = 1, ... , k, and so the proof of statement (a) is complete since $G(V_{p+1})$ through $G(V_k)$ form the nonclique connected components of G(V-S).

Condition (b): This follows immediately from Corollary 3.1 since $S \subseteq S^*$.

This completes the proof.

By construction, each of the p cliques $G(V_1)$ through $G(V_p)$ is a connected component of $G(V-S^*)$, and so by letting $G(\mathbf{V}_1)$ through $G(\mathbf{V}_q)$ denote the remaining connected components of $G(V-S^*)$ it follows that the set of vertex sets Π^* defined by

$$\Pi^* = (V_1, \ldots, V_p, \mathbf{V}_1, \ldots, \mathbf{V}_q, S^*)$$

is a vertex partition in G and furthermore, the graph G_{Π^*} with respect to Π^* has the star-shaped form shown in figure 9 where each leaf vertex corresponds to a clique in G. Note that if we let r = p+q and $V_i = \mathbf{V}_{p+i}$, for i = 1, ... , q then we have the vertex partition Π^* introduced in the introduction of the work.

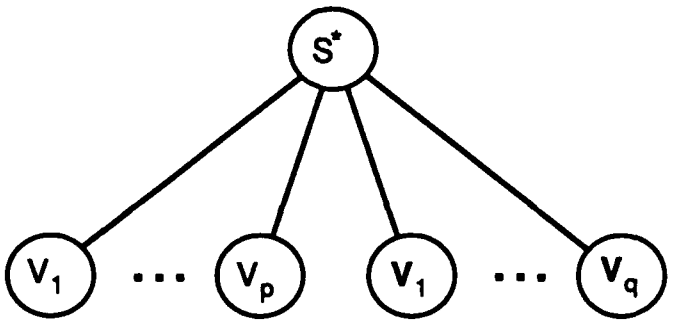

Figure 9. The graph G_{Π^*} with respect to the partition Π^*.

To establish the matrix interpretation of the partition Π^*, let $\mathbf{A}_{ii}$ be a square block in M such that its rows correspond to the vertices in the set $\mathbf{V}_i$ for i =1, ... , q. Also, let D_{11} and D_{22} be square blocks such that the rows of D_{11} correspond to the vertices in the set $(S_{p+1} \cup \ldots \cup S_k)$, while the rows of D_{22} correspond to the vertices in S. Then by the structures of the star-shaped graphs in figures 7 through 9, there exists a permutation matrix P such that PMP^T is a block bordered diagonal matrix of the following form

$$PMP^T = \begin{bmatrix} A_{11} & & & & & B_{12} \\ & \ddots & & & & \vdots \\ & & A_{pp} & & & B_{p2} \\ & & & A_{11} & B_{11} & B_{12} \\ & & & & \ddots \; \vdots & \vdots \\ & & & & A_{qq} \; B_{q1} & B_{q2} \\ & & & B_{11}^T \cdots B_{q1}^T & D_{11} & D_{12} \\ B_{12}^T \cdots & B_{p2}^T & B_{12}^T \cdots & B_{q2}^T & D_{12}^T & D_{22} \end{bmatrix} \qquad (12)$$

Since each leaf vertex of G_{Π^*} corresponds to a clique in G, all diagonal block in the leading (p+q) by (p+q) block diagonal matrix in PMP^T are full matrices.

There are several ways in which the block matrix PMP^T in (12) can be used in the solution of linear symmetric systems of equations. One approach is to choose the p-by-p leading block diagonal matrix in PMP^T as the pivot block A. This way, the Schur complement $D - X^TX$ will be a block bordered diagonal matrix with a q-by-q leading block diagonal matrix. Another way is to choose the (p+q)-by-(p+q) leading block diagonal matrix in PMP^T as the pivot block A. The choice between these two alternative pivot blocks and others will depend on the parameters p and q, the number of available processors in a parallel machine, as well as the sizes of the diagonal blocks in the pivot block.

11. SOLUTION STRATEGY USING STRUCTURED MATRIX

For any permutation matrix P, the system of equations Mx = b is equivalent to the following system of equations

$$(PMP^T)(P^Tx) = (Pb). \tag{13}$$

Consider this system where PMP^T has been factored into the block product form in equation (7), and the vectors P^Tx and Pb have been partitioned conformably into the direct sums of y and z, and f and h, respectively. The system of equations (13) then becomes

$$\begin{bmatrix} U & X \\ 0 & D-X^TX \end{bmatrix}\begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} w \\ h-X^Tw \end{bmatrix}, \tag{14-a}$$

where

$$U^Tw = f. \tag{14-b}$$

So by combining equations (8) and (14) we obtain the following systems of equations

$$U^T[\,X\;\;w\,] = [\,B\;\;f\,], \tag{15-a}$$
$$[\,D - X^TX\,]\,z = h - X^Tw, \tag{15-b}$$
$$U\,y = w - Xz. \tag{15-c}$$

The system of equations (15-a) is a lower triangular system with multiple right-hand sides. The solution of this system provides the block X and the vector w. Subsequently, the solution of the system of equations (15-b) provides the vector z, which is part of the unknown vector x. Lastly, the solution of the system of equations (15-c) yields the remaining part y of the unknown vector x.

The solving of a triangular system with multiple right-hand sides in equation (15-a) and the rank-k matrix update $D - X^TX$ in equation (15-b) form two of the three fundamental linear algebra operations that are referred to as Level 3 operations (Anderson et al., 1992). The other Level 3 operation concerns matrix-matrix multiplication. For many high-performance computer architectures, and especially those with complex memory hierarchies, Level 3 operations are extremely desirable from a computation standpoint since they require the loading and storing of $O(n^2)$ data while involving $O(n^3)$ computations. This favourable computation-to-communication ratio significantly reduces traffic between various memory hierarchies and consequently leads to a methodology for the development of fast algorithms on high-performance computers. In contrast to Level 3 operations, the other linear algebra operations such as vector-vector operations (Level 1) and matrix-vector operations (Level 2) require the loading and storing of O(n) data and O(n) computations or

$O(n^2)$ data and $O(n^2)$ computations, respectively. This means that Level 1 and Level 2 linear algebra operations are unable to accomplish the high ratio of computation to communication attained by Level 3 operations.

The problem of factoring a matrix into the product of lower and upper triangular matrices involves a mix of Level 1 and Level 2 linear algebra operations (Anderson *et al.*, 1992). Thus a key objective in factoring a matrix into the block product form in equation (7) is to decrease work involving Level 1 and Level 2 linear algebra operations and increase work that requires Level 3 operations.

Also, the block product form in equation (7) is well-suited for parallel computation especially when the pivot block A is a block diagonal martix. To see this, consider the case where PMP^T has the block bordered diagonal form in equation (11), the block X has been partitioned conformably into the form

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \cdot \\ \cdot \\ \cdot \\ X_k \end{bmatrix}, \tag{16}$$

and the vectors y, f, and w have been partitioned conformably into the direct sums of y_1 through y_k, f_1 through f_k, and w_1 through w_k, respectively. Then the block product in equation (6) takes the form

$$A_{ii} = U_{ii}^T U_{ii} \qquad i = 1, \ldots, k, \tag{17}$$

and the system of equations (15) becomes

$$U_{ii}^T [\, X_i \;\; w_i \,] = [\, B_i \;\; f_i \,], \tag{18-a}$$

$$[D - \Sigma_{i=1}^{k} X_i^T X_i]\, z = h - \Sigma_{i=1}^{k} X_i^T w_i, \tag{18-b}$$

$$U_{ii}\, y_i = w_i - X_i\, z, \qquad i = 1, \ldots, k. \tag{18-c}$$

The operations in equations (17) and (18) can be tailored to take full advantage of parallel machines. To begin with, the k diagonal blocks A_{11} through A_{kk} of the pivot block A are factored in parallel to produce k upper triangular matrices U_{11} through U_{kk}. Then, a total of k triangular systems with multiple right-hand sides are solved in parallel to produce the blocks X_1 through X_k and the vectors w_1 through w_k. Next, the k matrix-matrix multiples $X_1^T [X_1 \;\; w_1]$ through $X_k^T [\, X_k \, w_k \,]$ are computed in parallel to produce the Schur complement $D - X^TX$ and the right hand vector $h - X^Tw$ in equation (18-b). The solution of equation (18-b) produces the vector z. Subsequently, the vector y is obtained using equation (18-c) by solving another k triangular systems in parallel.

12. RECURSIVE EXPLOITATION OF PARALLELISM IN SPARSE PROBLEMS

Many large complex scientific and engineering problems in government and industry give rise to block bordered diagonal matrices of the form PMP^T in equation (11) where the Schur complement $D-B^TA^{-1}B$ of A in PMP^T is also a large sparse matrix. One such problem concerns the use of barrier methods or interior-point methods for solving linear programs and nonlinear programs with linear equality constraints. The application of barrier or interior-point methods to these optimization problems gives rise to a sparse symmetric system of equations called the KKT system (Gill et al., 1991; Gill et al., 1992), in which the coefficient matrix K has the following 2-by-2 block form

$$K = \begin{bmatrix} H & A^T \\ A & D \end{bmatrix},$$

where H and A are both sparse matrices and D is either a zero matrix or a strictly negative diagonal matrix with small diagonal entries.

For general nonlinear programs, the leading block H in the K matrix has no specific sparsity structure. For linear programs, however, H is a positive diagonal matrix, which means that the coefficient matrix K in the KKT system is a bordered diagonal matrix and thus a special case of the block bordered diagonal matrix PMP^T.

Experimental results obtained from the application of the parallelization tool roadmap to a collection of linear programs* has shown that the Schur complement $D-AH^{-1}A^T$ of H in K is usually a sparse matrix (Kevorkian, in preparation-b). Moreover, further applications of roadmap to the Schur complements in the linear programs suggests worthwhile opportunities for parallelism in these large sparse matrices.

To exploit parallelism in a Schur complement, we must first compute the fill-in resulting from the use of the pivot block in PMP^T. The subject of computing fill-in has been an area of active research in the seventies and eighties (Rose, Tarjan, and Lueker, 1976; George and Liu, 1981; Yannakakis, 1981; Tarjan and Yannakakis, 1984). Among the many methods developed over the years, the linear-time algorithm devised by Tarjan and Yannakakis (1984) is the most efficient and easy to implement.

Currently we are working on a recursive version of roadmap (Kevorkian, in preparation-a) in which all fill-in computations are done using the Tarjan and Yannakakis algorithm. This recursive version of roadmap exploits parallelism in the original matrix as well as subsequent Schur complements until no further parallelism remains to exploit.

---

*Saunders, M. A., and W. Murray, private correspondence, 1992.

In the process of developing this extension of roadmap, we have come across two worthwhile properties of semi-interior cliques. One property relates to speeding up fill-in computations while the other concerns the computation of interior cliques in an undirected graph.

Lemma 3. Let G(U) be any semi-interior clique in G = (V, E), and let α be any ordering of the vertices in U. Then for any vertex u in U, the following equality holds.

$$F_G\alpha = def_G u.$$

Proof. If |U| = 1, there is nothing to prove since α consists of the single vertex u. Suppose |U| > 1. If the clique G(U) is interior, we have $F_G\alpha = \varnothing$ and so the assertion must hold. Suppose the clique G(U) is strictly semi-interior and let $G^* = (V, E^*)$ be the complement of G. Let v and w be any two distinct vertices adjacent to u. If both v and w are in U, then the pair (v, w) is an edge in E since G(U) is a clique. If v is in U and w is in $N_G U$, then (v, w) is an edge in E since every vertex in U is adjacent to all vertices in $N_G U$. Suppose v and w are both in $N_G U$. Then the edge (v, w) is either in E or E^* and so we get

$$def_G u = E^*(N_G U)$$

since every vertex in $N_G U$ is adjacent to vertex u.

By construction, the graph $(N_G U, E(N_G U) \cup E^*(N_G U))$ is a clique and so if we let E' be the set of edges defined by $E' = E \cup E^*(N_G U)$, then by Corollary 4.1 it follows that G(U) is an interior clique in the graph G' = (V, E'). Consequently, by Theorem 2 we get $F_G\alpha = E' - E = E^*(N_G U)$ and so we get the desired relation $F_G\alpha = def_G u$.

By Lemma 3, the application of the Tarjan and Yannakakis algorithm to any one vertex u in a semi-interior clique G(U) produces all fill-in resulting from the application of the algorithm to every vertex in U. Computations in (Kevorkian, in preparation-a) are thus speeded up by applying the Tarjan and Yannakakis algorithm to a single vertex in G(U) and ignoring all other vertices for all semi-interior cliques.

Our second result pertains to the computation of all interior and strictly semi-interior cliques in an undirected graph.

Lemma 4. Let α be any ordering of the vertex set V and let (v, w) be any edge in the set of edges $F_G\alpha$. Then every semi-interior clique of G with a starting vertex adjacent to both v and w is strictly semi-interior.

Proof. Let G(U) be any semi-interior clique with a starting vertex r adjacent to both v and w. Then by Lemma 3 both v and w are vertices in the neighborhood $N_G U$ of U and so (v, w) is an edge of the complement G^* of G. Thus the subgraph induced by the vertex set $N_G U$ is not a clique and so G(U) is a strictly semi-interior clique. This completes the proof.

By construction, any semi-interior clique that is not strictly semi-interior is an interior clique, and so by Lemma 4 we are able to compute all interior and strictly

semi-interior cliques in an undirected graph. Details on the implementation of Lemmas 3 and 4 will be covered in (Kevorkian, in preparation-a).

13. DISTRIBUTION OF WORKLOAD ACROSS PROCESSORS

An important concern in parallel computation is the proper distribution of workload across the processors of a parallel architecture computer. Ideally, a workload distribution is sought in which each processor has about the same amount of workload and in which the interprocessor communications are small. Unfortunately, the computation of such an idealistic workload distribution is extremely difficult in practice, especially for problems involving large sparse symmetric matrices with irregular structures.

Given a vertex partition $\Pi^* = (V_1, V_2, \ldots, V_r, S^*)$ computed by the parallelization tool roadmap, the sizes of the vertex sets V_1 through V_r in Π^* dictate the workloads that have to be distributed across the processors of the parallel machine. Now by the construction of the vertex partition Π^*, no vertex in any element V_i in Π^* is adjacent to a vertex in another element V_j. This means that the interprocessor communications between the r workloads resulting from the use of the vertex partition Π^* are simply absent. However, since our vertex partition algorithm does not provide any control over granularity of parallelism, the amounts of workloads distributed across the various processors may vary appreciably and thus degrade performance.

In subsequent developments we discuss ways for bringing about some control over the granularity of parallelism needed to adapt to varying numbers of processors. These discussions are intended to be preliminary. More detailed coverage of this topic including implementations and applications to real-world Navy problems will be reported in the near future.

Let PMP^T be a block bordered diagonal matrix induced by the vertex partition Π^*, and suppose we are given a parallel architecture computer in which nproc denotes the number of processors available for the solution of our particular sparse symmetric system of equations. Since r denotes the number of parallel regions produced by the vertex partition Π^*, one of the following two cases must hold.

Case 1. $r >$ nproc. Then we arbitrarily distribute nproc of the total r workloads across the nproc processors, and continue assigning a workload from the remaining r - nproc workloads to each processor that completes its designated task. This method should work reasonably well for problems in which none of the r workloads is significantly larger than the remaining workloads.

Case 2. $r \leq$ nproc. Let s be the positive integer defined by

$$s = \min_{1 \leq i \leq r} |V_i| ,$$

and let A_{ii} be any of the r diagonal blocks in the pivot block A of PMP^T. Now let α_{ii} be the leading s by s principal submatrix in A_{ii}. If $|V_i| = s$, then $\alpha_{ii} = A_{ii}$. Otherwise, we can write A_{ii} in the following 2-by-2 block form

$$A_{ii} = \begin{bmatrix} \alpha_{ii} & \beta_{ii} \\ \beta_{ii}^T & \delta_{ii} \end{bmatrix}.$$

For this discussion, we assume that the original matrix M is positive definite, which means that the s-by-s leading block α_{ii} in the 2-by-2 block matrix A_{ii} is nonsingular.

With this partitioning of the r diagonal blocks in the pivot block A, we are able to compute r upper triangular matrices U_{11} through U_{rr} across r processors using the following factorization.

$$\alpha_{ii} = U_{ii}^T U_{ii}, \qquad i = 1, \ldots, r. \quad (19)$$

The workload across the r processors of the parallel machine will be perfectly balanced during this factorization step since the blocks α_{11} through α_{rr} are full matrices of the same size.

Let B_i be any block in the border of PMP^T, and suppose we partition B_i into the 2-by-1 block form

$$B_i = \begin{bmatrix} B_{i_1} \\ B_{i_2} \end{bmatrix}$$

such that the block at the top has s rows. Now define the 2-by-2 block matrix

$$M_i = \begin{bmatrix} \alpha_{ii} & \mathbf{B}_i \\ \mathbf{B}_i^T & \mathbf{D}_i \end{bmatrix}.$$

where

$$\mathbf{B}_i = [\, \beta_{ii} \;\; B_{i_1} \,]$$

and

$$\mathbf{D}_i = \begin{bmatrix} \delta_{ii} & B_{i_2} \\ B_{i_2}^T & D \end{bmatrix}.$$

Then by setting the blocks A, B, and D in the 2-by-2 block matrix PMP^T in equation (5) to α_{ii}, $\mathbf{B}_i$ and $\mathbf{D}_i$, respectively, the block factorization in equation (7) yields

$$M_i = \begin{bmatrix} U_{ii}^T & 0 \\ X_i^T & I \end{bmatrix} \begin{bmatrix} U_{ii} & X_i \\ 0 & \mathbf{D}_i - X_i^T X_i \end{bmatrix}, \quad (20)$$

where the block $\mathbf{X}_i$ is the solution of the following triangular system with multiple right-hand sides

$$\mathbf{U}_{ii}^T \mathbf{X}_i = \mathbf{B}_i. \qquad i = 1, \ldots, r. \quad (21)$$

By construction, the block $\mathbf{B}_i$ in equation (21) has s rows and $|V_i| - s + |S^*|$ columns, and so each of the r systems in equation (21) is a triangular system with s equations. The right-hand sides of these r systems of equations, however, will vary in size whenever two or more diagonal blocks in the pivot block A have unequal dimensions. Therefore, to devise a scheme in which each processor solves a triangular system with s equations and about equal number of right-hand sides, we introduce the integer K defined by

$$K = \Sigma_{i=1}^{r} (|V_i| - s + |S^*|).$$

Then one of the follcwing two subcases must hold.

Subcase 2.1. K < nproc. Then the system of equations in equation (21) gives rise to K triangular systems of equations, each consisting of s equations and a single right-hand side. These triangular systems may be solved in parallel on K of the nproc processors of the parallel architecture machine. The remaining nproc - K processors are left idle at this stage of the computation. Evidently, this subcase identifies an instance where the problem to be solved is inherently small relative to the size of the given parallel architecture machine.

Subcase 2.2. K ≥ nproc. Let N be the positive parameter defined by

$$N = K / nproc.$$

For clarity of presentation, we will assume that N is an integer. Otherwise, we let N be the smallest integer greater than or equal to K / nproc.

Given the positive integer N we partition the blocks $\mathbf{X}_i$ and $\mathbf{B}_i$ in equation (21) into 1-by-λ_i block matrices of the form

$$\mathbf{X}_i = [\, \mathbf{X}_{i_1} \; \mathbf{X}_{i_2} \; \ldots \; \mathbf{X}_{i_{\lambda_i}} \,] \qquad (22\text{-a})$$

and

$$\mathbf{B}_i = [\, \mathbf{B}_{i_1} \; \mathbf{B}_{i_2} \; \ldots \; \mathbf{B}_{i_{\lambda_i}} \,], \qquad (22\text{-b})$$

such that the number of columns in each of the leading λ_i -1 blocks in both $\mathbf{X}_i$ and $\mathbf{B}_i$ is equal to N, and the number of columns in each of the two end blocks is less than or equal to N. Again, for clarity of presentation we will assume that all blocks in both $\mathbf{X}_i$ and $\mathbf{B}_i$ have N columns.

The partitioning of X_i and B_i into the block forms given in equation (22) decomposes the solution of the r triangular systems in equation (21) to the solution of nproc triangular systems of the form

$$U_{ii}^T X_{i_j} = B_{i_j}, \qquad i = 1, \ldots, r\,;\, j = 1, \ldots, \lambda_i, \quad (23)$$

in which each triangular system comprises s equations and N right-hand sides. These nproc triangular systems are subsequently solved in parallel on the nproc processors of the parallel machine. Clearly, all these nproc computations require the same amount of work since the r lower triangular matrices in equation (23) are of the same size and each triangular system has the same number of right-hand sides.

Besides the solution of the nproc triangular systems given in equation (23), we must also compute the r matrix-matrix multiplies defined by

$$Y_i = X_i^T X_i, \qquad i = 1, \ldots, r, \quad (24)$$

since the Schur complements $D_i - X_i^T X_i$ for $i = 1, \ldots, r$ are required for the completion of the solution process.

In the general case, the blocks X_1 through X_r have different dimensions, and so the computation presented in equation (24) does not lead to a proper workload distribution across the processors. However, if we combine relations (22-a) and (24) then the problem of computing the products Y_1 through Y_r is decomposed into a large numebr of matrix-matrix multiples such that each of these matrix-matrix multiplies involves the same amount of workload. This way we are able to process equal amounts of work on different processors of the parallel machine as they become available.

Since the Schur complement $D_i - X_i^T X_i$ satisfies the following equality

$$D_i - X_i^T X_i = D_i - B_i^T \alpha_{ii}^{-1} B_i,$$

at the completion of the processing of the workloads defined in equations (23) and (24) we obtain

$$D_i - X_i^T X_i = \begin{bmatrix} \delta_{ii} - \beta_{ii}^T \alpha_{ii}^{-1} \beta_{ii} & B_{i_2} - \beta_{ii}^T \alpha_{ii}^{-1} B_{i_1} \\ B_{i_2}^T - B_{i_1}^T \alpha_{ii}^{-1} \beta_{ii} & D - B_{i_1}^T \alpha_{ii}^{-1} B_{i_1} \end{bmatrix}.$$

Consequently, we set

$$A_{ii} = \delta_{ii} - \beta_{ii}^T \alpha_{ii}^{-1} \beta_{ii},$$
$$B_i = B_{i_2} - \beta_{ii}^T \alpha_{ii}^{-1} B_{i_1},$$

and

$$D = D - B_{i_1}^T \alpha_{ii}^{-1} B_{i_1},$$

and repeat the procedure outlined in Case 2 until each of the original diagonal blocks A_{ii} in the block bordered diagonal matrix PMP^T is entirely factored.

In connection to the repeated application of Case 2, we would like to make one final comment. Since the integer s denotes the size of the pivot block α_{ii} in equation (19), one can compute the upper and lower triangular factors in equation (19) as soon as the first s columns of the blocks $\mathbf{X}_1$ through $\mathbf{X}_r$ are computed. This way we are able to bring about more parallel processing in our solution strategy by doing parts of the computations in equations (19) and (23) in parallel.

14. AN ILLUSTRATION OF ROADMAP IMPLEMENTATION

We will demonstrate the program roadmap reported in Kevorkian (in preparation-a) using the 21-by-21 structurally symmetric matric M shown in figure 10. The "x"s denote nonzero entries and the blanks denote zeros. As can be noted, the sparsity structure of matrix M is quite irregular. This was purposely done to illustrate generality as well as the classification of cliques presented earlier.

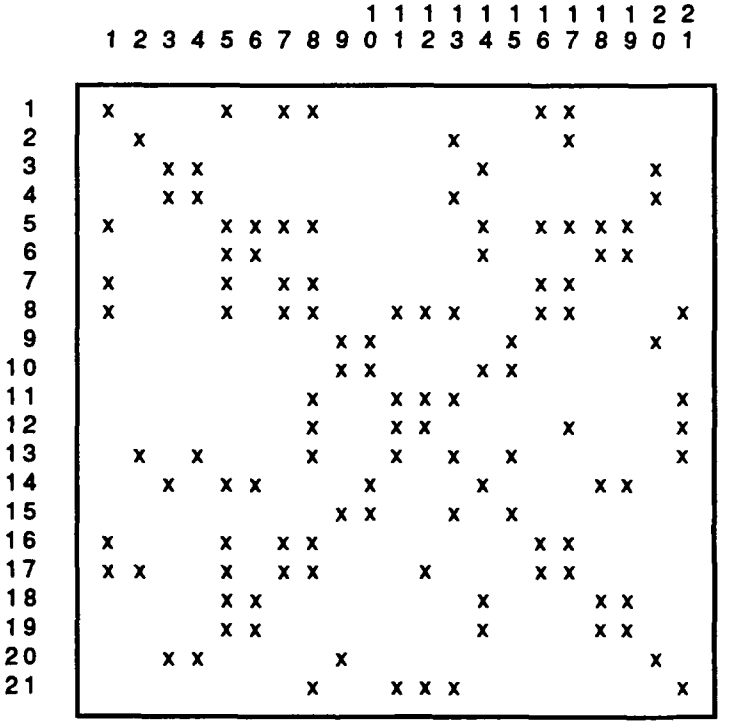

Figure 10. Structurally symmetric matrix M.

Suppose G = (V, E) is the undirected graph of matrix M. The objective of procedure search is to compute in G the set of vertices S. At the completion of search, each of the five vertices v_5, v_8, v_{13}, v_{14}, and v_{17} has separator number equal to 1. All other vertices in V have separator numbers equal to zero. Thus if we were to compute S explicitly, then we will obtain the array

$$S = [\,5, 8, 13, 14, 17].$$

Since S is not empty, the program roadmap calls procedure dfs. For clarity of presentation, let us first execute roadmap with the call to procedure cliques (in classify) excluded. Then at the completion of roadmap we obtain the following three arrays:

$$TYPE = [\,-2, -2, 0, -2, 3\,],$$

$$QUEUE = [\,1, 7, 16, 2, 3, 4, 20, 9, 10, 15, 6, 18, 19, 11, 12, 21\,],$$

and

$$IQUEUE = [\,1, 4, 5, 11, 14, 17\,].$$

Since TYPE is an array of length 5, the subgraph of G induced by the set of vertices V-S has five connected components. These five connected components are placed on the single array QUEUE one at a time in the order they get computed. The first five integers on the array IQUEUE are the pointers to the starting vertices of these five connected components. The first integer 1 on IQUEUE points to the starting vertex v_1 of the first connected component computed in dfs, the second integer 4 on IQUEUE points to the starting vertex v_2 of the second connected component, up to the fifth integer 14 on IQUEUE, which points to the starting vertex v_{11} of the fifth and last connected component computed in dfs. The sixth and last integer placed on the array IQUEUE is the pointer to the end of the array QUEUE. The overall relationship between the three single arrays TYPE, QUEUE, and IQUEUE is summarized in figure 11.

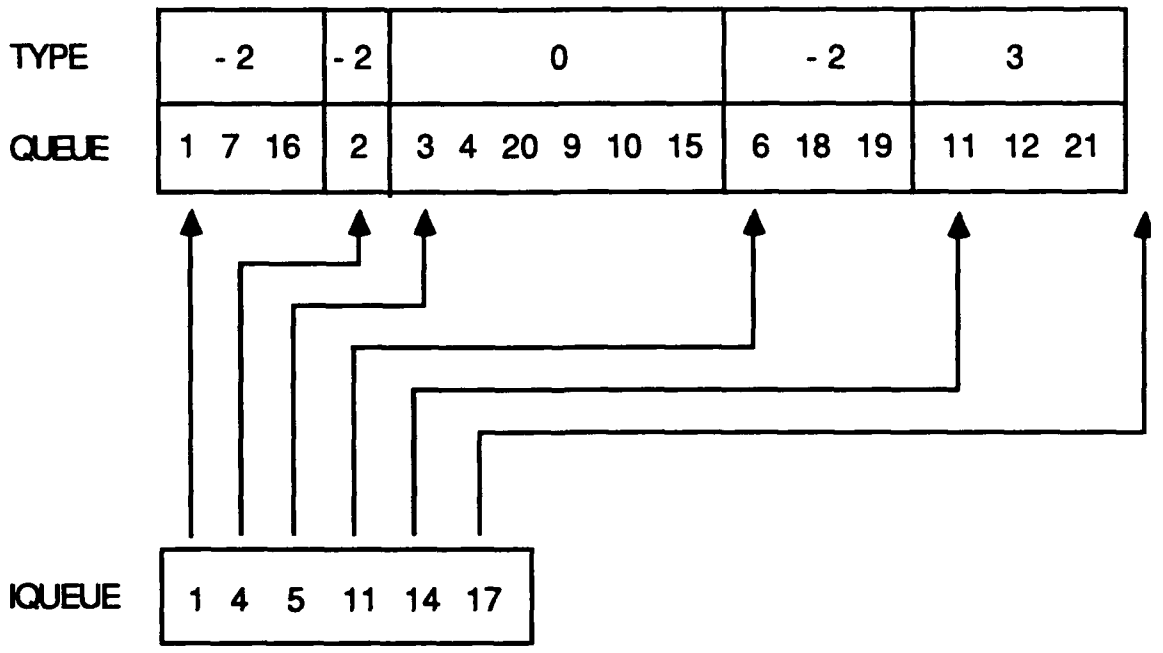

Figure 11. Overall relationship between the arrays TYPE, QUEUE, and IQUEUE.

Let $G(V_1)$, $G(V_2)$ through $G(V_5)$ denote the five connected components placed on array QUEUE in that order. Then by the configuration in figure 11 we have

$$V_1 = \{ v_1, v_7, v_{16} \},$$
$$V_2 = \{ v_2 \},$$
$$V_3 = \{ v_3, v_4, v_{20}, v_9, v_{10}, v_{15} \},$$
$$V_4 = \{ v_6, v_{18}, v_{19} \},$$
$$V_5 = \{ v_{11}, v_{12}, v_{21} \}.$$

These five vertex sets together with the set of vertices S computed in procedure search provide a vertex partition Π defined by

$$\Pi = (V_1, V_2, V_3, V_4, V_5, S) .$$

The star-shaped graph G_Π with respect to the vertex partition Π is shown in figure 12.

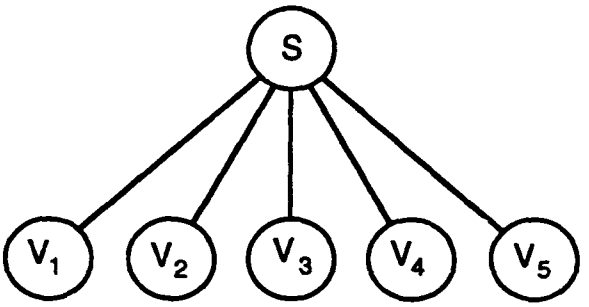

Figure 12. The graph G_Π with respect to vertex partition Π.

To categorize the leaf vertices in the graph G_Π, we need to examine the array TYPE computed in roadmap. Since TYPE [3] = 0, the third connected component $G(V_3)$ placed on array QUEUE is not a clique. The other four connected components placed on array QUEUE are cliques since none of the other four integers on array TYPE is zero. Among these four cliques, each of the three cliques $G(V_1)$, $G(V_2)$, and $G(V_4)$ is semi-interior (type C_1 or C_2) since TYPE[1] = TYPE[2] = TYPE[4] = - 2. The other clique, $G(V_5)$, placed on array QUEUE is not semi-interior (type C_3 or C_4) since TYPE[5] = 3. To verify these findings, consider the graph G of matrix M shown in figure 13. The neigborhoods of the four vertex sets V_1, V_2, V_4, and V_5 are, respectively, $N_G V_1 = \{v_5, v_8, v_{17}\}$, $N_G V_2 = \{v_{13}, v_{17}\}$, $N_G V_4 = \{v_5, v_{14}\}$, and $N_G V_5 = \{v_8, v_{13}, v_{17}\}$. As can be seen from the graph in figure 13, each vertex in the sets V_1, V_2, and V_4 is adjacent to all vertices in the neighborhoods $N_G V_1$, $N_G V_2$, and $N_G V_4$, respectively, while the vertices v_{12} and v_{21} in the set V_5 are not adjacent to all vertices in the neighborhood $N_G V_5$.

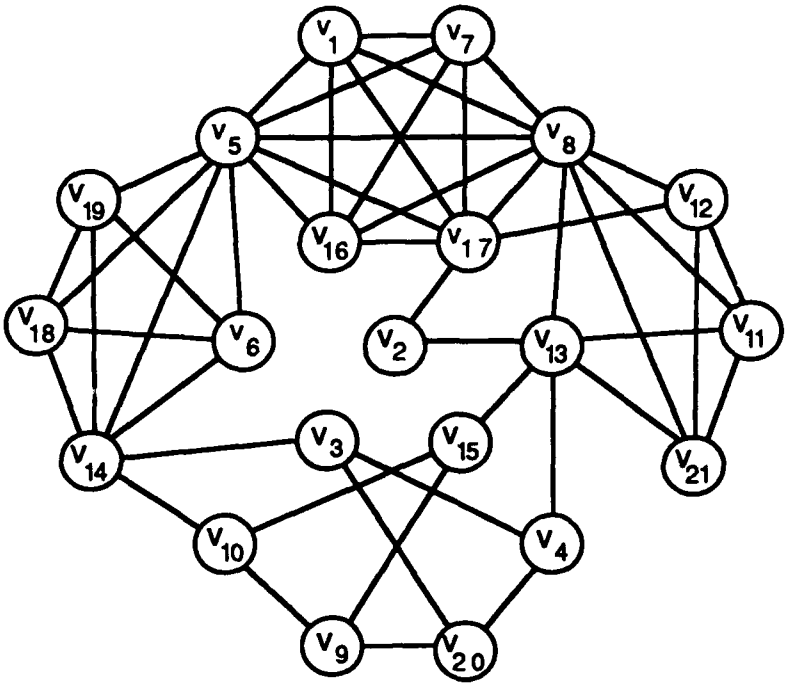

Figure 13. The graph G = (V, E) of matrix M.

Also, among the three semi-interior cliques placed on array QUEUE, $G(V_1)$, and $G(V_4)$ are interior cliques since the subgraphs induced by the neighborhoods $N_G V_1$ and $N_G V_4$ are cliques. However, the third semi-interior clique, $G(V_2)$, is strictly semi-interior since the subgraph induced by the neighborhood $N_G V_2$ is not a clique. The categorization of the five connected components of G(V-S) into cliques and noncliques and the subsequent classification of the clique connected components of G(V-S) into semi-interior and nonsemi-interior cliques divides the five leaf vertices of G_Π into three disjunct groups as shown in figure 14. Further classification of semi-interior cliques into interior and strictly semi-interior cliques will be covered in Kevorkian (in preparation-a).

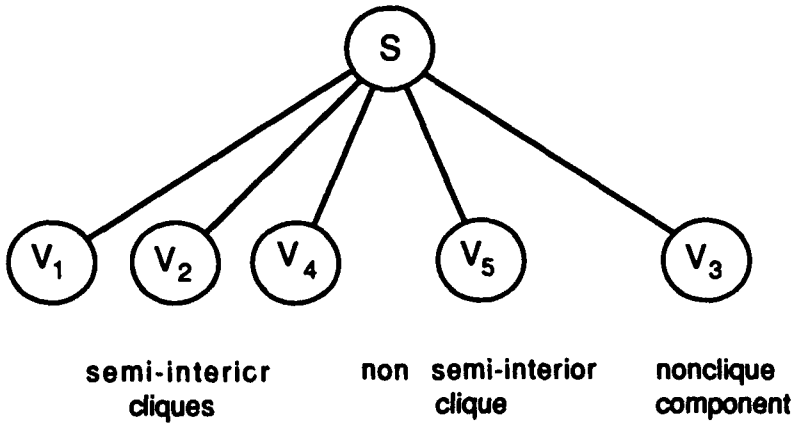

Figure 14. Grouping of five leaf vertices of G_Π using the array TYPE.

To complete our illustration, we apply program roadmap to matrix M, now with the call to procedure cliques in classify included.

At the first two calls to procedure classify in dfs, roadmap returns to dfs without calling procedure cliques since both $G(V_1)$ and $G(V_2)$ are cliques. However, at the third call to classify, procedure cliques is invoked since $G(V_3)$ is not a clique. This constitutes the last call to procedure cliques since the remaining two connected components of G(V-S) are cliques. At the completion of roadmap, the arrays QUEUE and IQUEUE are as follows:

$$QUEUE = [\ 1, 7, 16, 2, 3, 4, 20, 10, 15, 9, 6, 18, 19, 11, 12, 21\ ],$$

and

$$IQUEUE = [\ 1, 4, 5, -8, -10, 11, 14, 17\ ].$$

The array TYPE remains unchanged at the completion of this second application of roadmap.

Since the connected component $G(V_3)$ is not a clique, the integer 5 on IQUEUE (pointer to the starting vertex of $G(V_3)$) is also the pointer to the starting vertex of the first independent clique computed in connected component $G(V_3)$. The negative integer -8 is the pointer to the starting vertex of the second independent clique computed in $G(V_3)$. The negative integer -10 is the pointer to the starting vertex of the set of vertices S' defined in relation (9). The overall relationship between the three arrays TYPE, QUEUE, and IQUEUE is presented in figure 15.

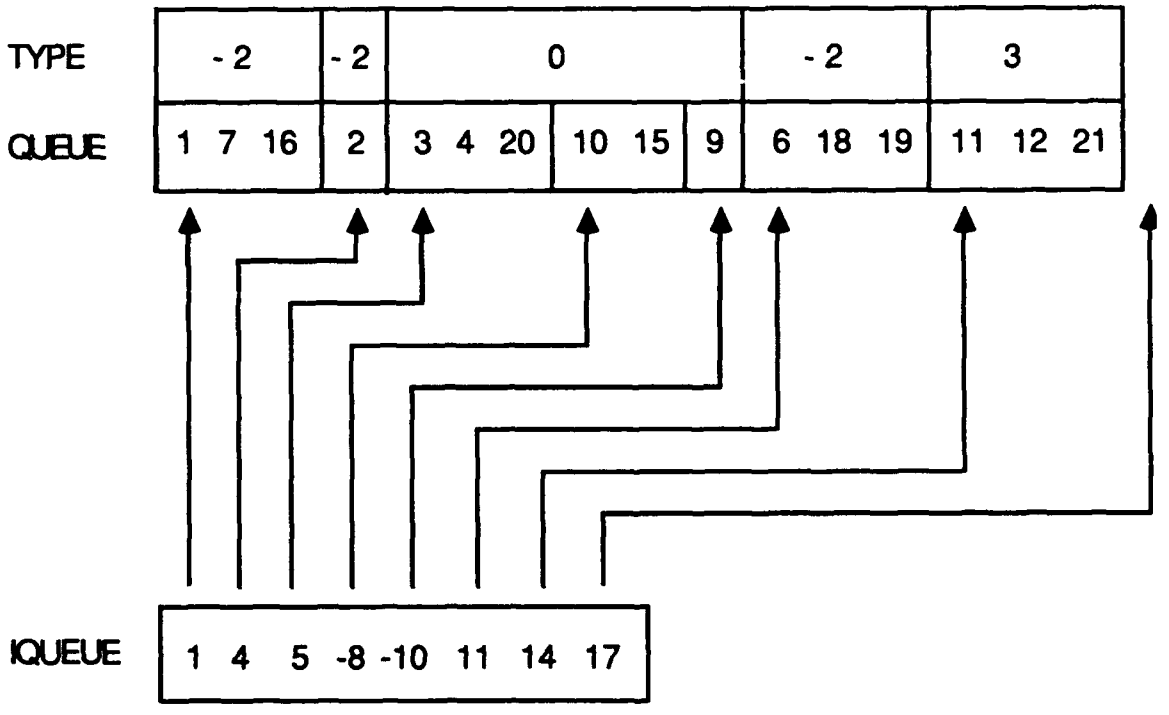

Figure 15. Arrays TYPE, QUEUE, and IQUEUE at the completion of roadmap.

The output of roadmap given in figure 15 provides the information leading to the final vertex partition Π^*. By the three pointers 5, -8, and -10 on array IQUEUE, the application of cliques to connected component $G(V_3)$ has produced a vertex partition Π' in $G(V_3)$ defined by

$$\Pi' = (U_1, U_2, S'),$$

where

$$U_1 = \{ v_3, v_4, v_{20} \},$$
$$U_2 = \{ v_{10}, v_{15} \},$$

and

$$S' = \{v_9\}.$$

By the conditions imposed in procedure cliques, $G(U_1)$ is a maximal clique in $G(V_3)$ and $G(U_2)$ is a maximal clique in subgraph of G induced by the set of vertices obtained from V_3 by deleting the vertices in U_1 and its neighborhood. The vertex set S' separates $G(U_1)$ and $G(U_2)$ into two independent cliques. Combination of the vertex partitions Π and Π' provides another vertex partition Π'' defined by

$$\Pi'' = (V_1, V_2, U_1, U_2, S', V_4, V_5, S).$$

Consequently, if we let S^* be the set of vertices defined by

$$S^* = S \cup S',$$

then by Theorem 6 the tuple of seven vertex sets

$$\Pi^* = (V_1, V_2, V_4, V_5, U_1, U_2, S^*)$$

forms a vertex partition such that the graph G_{Π^*} with respect to Π^* is the star-shaped graph shown in figure 16.

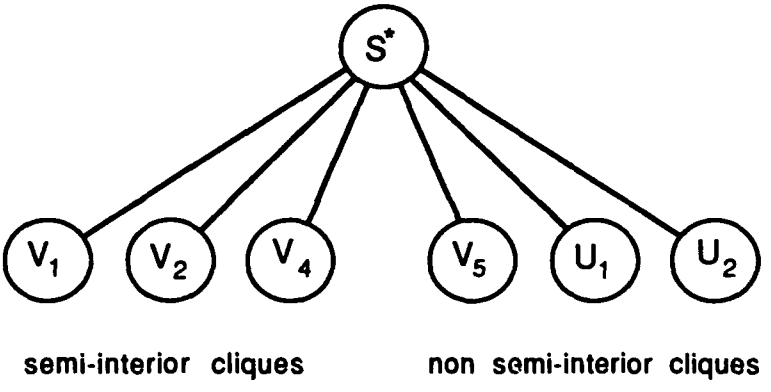

Figure 16. The graph G_{Π^*} with respect to vertex partition Π^*.

By Theorem 6, every connected component of G(V - S*) is a clique and so every leaf vertex of the graph G_{Π^*} corresponds to a clique in G. Also by Theorem 6, every interior clique in G is a connected component of G(V-S*), which means that every interior clique in G is represented as a leaf vertex in G_{Π^*}. The two interior cliques in G are $G(V_1)$ and $G(V_4)$, and both cliques are accounted for in the graph G_{Π^*}.

We conclude our application of the program roadmap to matrix M with a matrix interpretation of the graph G_{Π^*} in figure 16.

Suppose α is an ordering of the vertex set V obtained by picking vertices from the first through seventh element of the vertex partition Π^* in that order. Also, let P be a permutation matrix such that row i of matrix PM corresponds to ith element of the vertex ordering α. Then for the case that the ordering α is given by the 21-tuple

$$\alpha = (1, 7, 16, 2, 6, 18, 19, 11, 12, 21, 3, 4, 20, 10, 15, 9, 5, 8, 13, 14, 17),$$

the matrix PMP^T takes the 7-by-7 block bordered diagonal form shown in figure 17.

	1	7	16	2	6	18	19	11	12	21	3	4	20	10	15	9	5	8	13	14	17
1	x	x	x														x	x			x
7	x	x	x														x	x			x
16	x	x	x														x	x			x
2				x															x		x
6					x	x	x										x			x	
18					x	x	x										x			x	
19					x	x	x										x			x	
11								x	x	x								x	x		
12								x	x	x								x			x
21								x	x	x								x	x		
3											x	x	x							x	
4											x	x	x						x		
20											x	x	x			x					
10														x	x	x				x	
15														x	x	x			x		
9													x	x	x	x					
5	x	x	x		x	x	x										x	x		x	x
8	x	x	x					x	x	x							x	x	x		x
13				x				x		x		x			x			x	x		
14					x	x	x				x			x			x			x	
17	x	x	x	x					x								x	x			x

Figure 17. The 7-by-7 block bordered diagonal matrix PMP^T.

The pivot block A in this block bordered diagonal matrix is a 6-by-6 block diagonal matrix in which the ith diagonal block corresponds to the subgraph induced by the ith element of the vertex partition π^*. Therefore, each diagonal block in A is a full matrix since the subgraphs induced by the leading six elements of the vertex partition π^* are cliques. The general form of the block bordered diagonal matrix in figure 17 was established in relation (12).

Application of Theorem 5 to the block bordered diagonal matrix PMP^T leads to the following observations. First, by statement (a) we have $str(X_i) = str(B_i)$ and $str(D - X_i^T X_i) = str(D)$ for $i = 1, 3$, since the subgraphs induced by the first and third elements of the vertex partition π^* are interior cliques. Therefore no fill-in will occur in any part of matrix M if the first and third diagonal blocks of the pivot block A are factored symbolically. Second, by statement (b) we have $str(X_2) = str(B_2)$ and $str(D - X_2^T X_2) \neq str(D)$ since the subgraph induced by the second element of the vertex partition π^* is a strictly semi-interior clique. Thus no fill-in will occur in blocks X and X^T of the block product in (7) if the second diagonal block of A is factored symbolically. However, there will occur fill-in in the Schur complement $D - X^T X$ in locations (13, 17) and (17, 13) of the original matrix M since the pair (v_{13}, v_{17}) is not an edge in E and $N_G V_2 = \{v_{13}, v_{17}\}$. Third, by statements (c) and (d) we have $str(X_i) \neq str(B_i)$ for $i = 4, 5, 6$, since none of the subgraphs induced by the fourth through sixth elements of the vertex partition π^* is a semi-interior clique. With all these facts combined, the symbolic factorization of the block bordered diagonal matrix PMP^T into the block product form in (7) produces the fill-ins shown in figure 18.

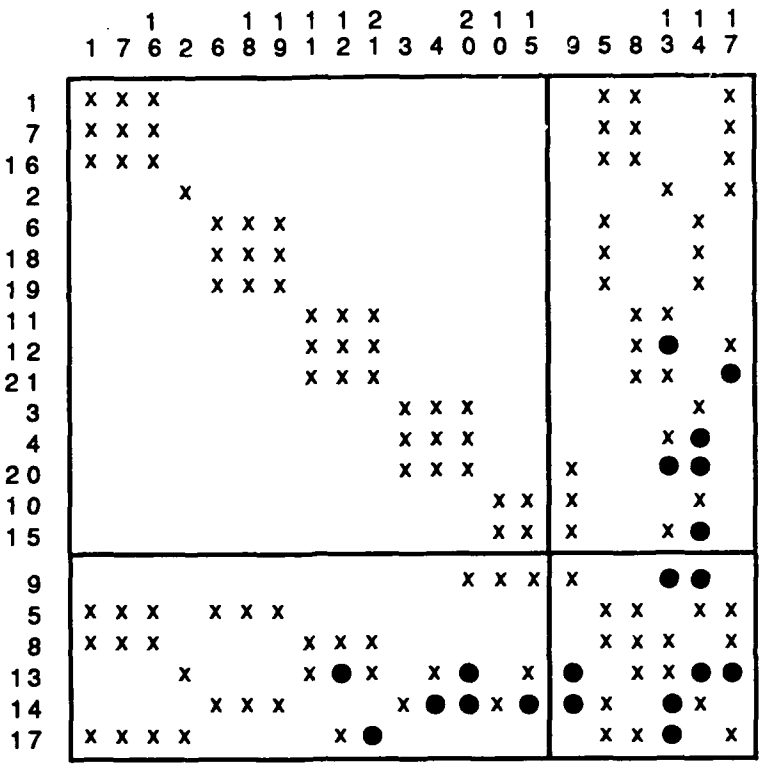

Figure 18. Block bordered diagonal matrix PMP^T with generated fill-in.

Experimental results obtained from the application of program roadmap to a standard set of test problems including the Harwell-Boeing and a collection of matrices arising from optimization problems will be reported in Kevorkian (in preparation-b).

15. CONCLUSIONS

We have presented an efficient linear-time vertex partition algorithm to decompose large sparse symmetric systems of equations into independently solvable smaller tasks for execution on different processors of a parallel architecture computer. The computationally independent tasks generated by the algorithm include all full principal submatrices that preserve sparsity in the process of symbolic factorization. This capability forms the most novel part of the presented algorithm.

We have extended the art of constructing block bordered diagonal forms of large sparse symmetric matrices by developing a method that applies to matrices with highly irregular sparsity structures and without any reliance on the expertise of modelers.

A complete implementation of a computer program incorporating the parallelization tool (using the linear algebra package Matlab) is covered in Kevorkian (1993).

16. REFERENCES

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1976. *The Design and Analysis of Computer Algorithms*, 3rd printing. Addison-Wesley, Reading, MA.

Anderson, E. et al. 1992. SIAM *LAPACK Users' Guide*. Philadelphia, PA.

Branin, F. H. 1975. "A Sparse Matrix Modification of Kron's Method of Piecewise Analysis," *Proceedings of the 1975 IEEE Int. Symp. Circuits and Syst.*, 1975, pp. 383-386.

Chan, T. F., R. Glowinsky, J. Periaux, and O. Widlund, eds. 1989. Proc. Second International Symposium on Domain Decomposition Methods, Jan. 1988, Los Angeles, CA., Society for Industrial and Applied Mathematics, Philadelphia, PA.

Chua, L. O., and L. K. Chen. 1976. "Diakoptic and Generalized Hybrid Analysis," *IEEE Trans. Circuits and Syst.*, CAS-23, pp. 694-705.

Dirac, G. A. 1961. "On Rigid Circuit Graphs," *Abhandlungen aus dem Mathematischen Seminar der Universitat Hamburg*, 25, pp. 70-76.

Duff, I. S., A. M. Erisman, and J. K. Reid. 1986. *Direct Methods for Sparse Matrices*, Oxford University Press, London.

George, A., and J. W-H Liu. 1981. *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Inc., Englewood Cliffs.

George, A., and J. W-H Liu. 1989. "The Evolution of the Minimum Degree Algorithm," SIAM Review, 31, pp. 1-19.

George, A., W. Poole, and R. Voight. 1978. "A Variant of Nested Dissection for Solving n by n Grid Problems," *SIAM J. Numer. Anal.*, 15, pp. 662-673.

Gill, P. E., W. Murray, D. B. Poncele'on, and M. A. Saunders. 1991. "Solving Reduced KKT Systems in Barrier Methods for Linear and Quadratic Programming," Technical Report SOL 91-7, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA.

Gill, P. E., W. Murray, D. B. Poncele'on, and M. A. Saunders. 1992. "Preconditioners for Indefinite Systems Arising in Optimization," SIAM J. Matrix Anal. Appl., 13, pp. 292-311.

Glowinsky, R., G. H. Golub, G. A, Meurant, and J. Periaux, eds. 1988. Proc. First International Symposium on Domain Decomposition Methods, Jan. 1987, Paris, France, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Heath, M. T., E. Ng, and B. W Peyton. 1991. "Parallel Algorithms for Sparse Linear Systems," SIAM Review, 33, pp. 420-460.

Kevorkian, A. K. 1993. (Mar). "Decomposition of Large Sparse Symmetric Systems for Parallel Computation. Part 2. Parallelization Tool Roadmap," NCCOSC/NRaD TR1601.

Kevorkian, A. K. (in preparation-a). "Decomposition of Large Sparse Symmetric Systems for Parallel Computation. Part 3. Recursive Version of Parallelization Tool Roadmap," NCCOSC/NRaD Technical Report in preparation.

Kevorkian, A. K. (in preparation-b). "Decomposition of Large Sparse Symmetric Systems for Parallel Computation. Part 4. Experimental Results Using Parallelization Tool Roadmap," NCCOSC/NRaD Technical Report in preparation.

Kron, G. 1958. *Diakoptics*, MacDonald, London.

Lekkerkerker, C. G., and J. Ch. Boland. 1962. "Representation of a Finite Graph by a Set of Intervals on the Real Line," *Polska Akademia Nauk Fundamenta Mathematicae*, LI, pp. 45-64.

The MathWorks. 1990. *Pro-Matlab User's Guide*, South Natick, MA.

Noor, A., H. Kamel, and R. Fulton. 1978. "Substructuring Techniques - Status and Projections," *Computers and Structures*, 8 , pp. 621-632.

Ogbuobiri, E. C., W. F. Tinney, and J. H. Walker. 1970. "Sparsity-directed Decomposition for Gaussian Elimination on Matrices," *IEEE Trans. on Power Apparatus and Systems*, PAS-80, pp. 141-150.

Parter, S. 1961. "The Use of Linear Graphs in Gauss Elimination." *SIAM Review*, 3, pp. 119-130.

Rabbat, N., A. Sangiovanni-Vincentelli, and H. Hsieh. 1979. "A Multilevel Newon Algorithm With Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain," *IEEE Trans. Circuits and Systems*, CAS-26, pp. 733-741.

Rose, D. J., R. E. Tarjan, and G. S. Lueker. 1976. "Algorithmic Aspects of Vertex Elimination on Graphs," *SIAM J. Comput.*, 5, pp. 266-283.

Tarjan, R. E. 1972. "Depth-First Search and Linear Graph Algorithms," *SIAM J. Comput.*, 1, pp. 146-160.

Tarjan, R. E., and M. Yannakakis. 1984. "Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs," *SIAM J. Comput.*, 13, pp. 566-579.

Yannakakis, M. 1981. "Computing the Minimum Fill-in is NP-Complete," *SIAM J. Alg. Disc. Meth.*, 2, pp. 77-79.

Zhang, X., R. H. Byrd, and R. B. Schnabel. 1992. "Parallel Methods for Solving Nonlinear Block Bordered Systems of Equations," *SIAM J. Sci. Stat. Comput.*, 13, pp. 841-859.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1993	Final

4. TITLE AND SUBTITLE

DECOMPOSITION OF LARGE SPARSE SYMMETRIC SYSTEMS FOR PARALLEL COMPUTATION
Part 1: Theoretical Foundations

5. FUNDING NUMBERS

AN: DN302038
PE: 0601152N
PROJ: ZW62

6. AUTHOR(S)

A. K. Kevorkian

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152-5001

8. PERFORMING ORGANIZATION REPORT NUMBER

NRaD TR 1572

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of the Chief of Naval Research
OCNR-10P
Arlington, VA 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Given a sparse symmetric matrix M, we develop a linear-time algorithm to construct in the undirected graph $G = (V, E)$ of M a vertex partition $\Pi^* = (V_1, V_2, \ldots, V_r, S^*)$ satisfying the following three properties. First, for any two distinct elements V_i and V_j of the partition, no vertex in V_i is adjacent to a vertex in V_j. Second, every element V_i of the partition induces a clique in G. Third, if A is a full principal submatrix of M such that the symbolic factorization of A does not produce fill-in, then the set of vertices in G corresponding to the rows of A is a subset of an element V_i of the partition. By the first two properties of the vertex partition Π^*, the solution of a sparse symmetric problem is reduced to the solution of smaller dense symmetric subproblems. In the case where M is a positive definite matrix, the first property allows us to factorize r dense symmetric blocks in parallel as well as solve in parallel r triangular systems with multiple right-hand sides. The third property is a means for computing an ordering that produces acceptably small fill-in.

14. SUBJECT TERMS

cliques
separators
fill-in
simplicial vertices
parallel computation
symbolic factorization

15. NUMBER OF PAGES

63

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(Include Area Code)*	21c. OFFICE SYMBOL
A. K. Kevorkian	(619) 553-2058	Code 7304

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 013	P. M. Reeves	(1)
Code 014	K. J. Campbell	(1)
Code 0141	A. Gordon	(1)
Code 02902	J. M. Baird	(1)
Code 0292	G. C. Pennoyer	(1)
Code 402	R. A. Wasilausky	(1)
Code 421	D. L. Conwell	(1)
Code 423	J. P. Schill	(1)
Code 542	F. P. Snyder	(1)
Code 5701	L. A. Parnell	(1)
Code 70	R. E. Shutters	(1)
Code 702	D. A. Hanna	(1)
Code 73	J. A. Roese	(1)
Code 7304	A. K. Kevorkian	(100)
Code 731	W. G. Thomson	(1)
Code 7601	K. N. Bromley	(1)
Code 78	R. H. Hearn	(1)
Code 782	R. Dukelow	(1)
Code 804	J. W. Rockway	(1)
Code 943	M. R. Blackburn	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(2)

Defense Technical Information Center
Alexandria, VA 22304–6145 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363–5100

Center for Naval Analyses
Alexandria, VA 22302–0268

Navy Acquisition, Research and Development
Information Center (NARDIC)
Washington, DC 20360–5000

GIDEP Operations Center
Corona, CA 91718–8000

NCCOSC Division Detachment
Warminster, PA 18974–5000

Office of Naval Research
Arlington, VA 22217–5000

Defense Advanced Research Projects Agency
Arlington, VA 22203–1714 (2)