Technical Report 1601
March 1993

Decomposition of Large Sparse Symmetric Systems for Parallel Computation

Part 2: Parallelization Tool Roadmap

A. K. Kevorkian

Approved for Public Release; Distribution is Unlimited

EXECUTIVE SUMMARY

OBJECTIVE

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

RESULTS

We have presented a detailed computer implementation of a combinatorial algorithm developed in part 1 (Kevorkian, 1993) 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. Also, we have presented a procedure that uses the output of the computer program to generate a block bordered diagonal form of matrix M that is well-suited for sparse block factorizations.

CONTENTS

EXECUTIVE SUMMARY i

1. INTRODUCTION 1

2. IMPLEMENTATION OF PARALLELIZATION TOOL ROADMAP 1

3. BLOCK BORDERED DIAGONAL FORMS USING ROADMAP 3

4. FUTURE WORKS 14

5. CONCLUSION 14

6. REFERENCES 14

FIGURES

1. Procedure roadmap 5

2. Procedure sprsdata 6

3. Procedure search 7

4. Procedure dfs 8

5. Function cmponent 9

6. Procedure classify 10

7. Procedure cliques 11

8. Procedure maxclq 12

9. Procedure bbdf 13

1. INTRODUCTION

The solution of large sparse linear symmetric systems of equations of the type

$$Mx = b$$

forms the most compute-intensive part in several of the Navy's fundamental grand challenge problems. These problems include ocean basin scale modeling, three-dimensional modeling of ocean acoustic propagation, fluid flow simulations, turbulent combustion, and structural design of navy vehicles, as well as linear and nonlinear constrained optimization problems known as linear and nonlinear programming. In most of these applications, there is significant parallelism hidden in the structure of the original problem. Therefore, any progress toward the efficient solution of a general grand challenge problem on a parallel architecture computer will require advanced computational tools that can exploit all hidden parallelism.

In this work we give a complete computer implementation of an algorithmic tool developed recently by Kevorkian (1993) for exploiting parallelism hidden in the sparsity structure of large sparse symmetric matrices with regular and irregular structures. The application of this automated parallelization tool to a general sparse symmetric system of equations decomposes the original problem into independently solvable smaller tasks for execution on different processors of a parallel architecture computer.

This report is organized as follows. In section 2, we present a computer implementation of the parallelization tool, called "roadmap" (Kevorkian, 1993), using the linear algebra package Matlab (MathWorks, 1990). In section 3, we present a procedure that uses the program roadmap to generate a permutation matrix P such that PMP^T is a block bordered diagonal matrix satisfying all three properties of the vertex partition computed in roadmap. In section 4, we discuss future works pertaining to experimental results obtained from the application of roadmap to a collection of test problems. Also, we discuss the development of a recursive version of roadmap that can exploit parallelism not only in the original problem, but also in all subsequent parts (Schur complements) that usually result from the solution of a sparse system of equations by Gaussian elimination.

2. IMPLEMENTATION OF PARALLELIZATION TOOL ROADMAP

In this section we present a complete computer implementation of roadmap by using the widely available linear algebra package Matlab (MathWorks, 1990).

The input to roadmap is an n-by-n structurally symmetric sparse matrix $M = [m_{ij}]$ with the data structure described as follows.

Suppose $G = (V, E)$ denotes the undirected graph of the n-by-n matrix M. Then the set V consists of n vertices v_1 through v_n, with v_i representing row i of matrix M. For convenience, let $v_i = i$, for $i = 1,\ldots,n$. Then we get $adj_G v_i = \{ j \mid m_{ij} \neq 0 \text{ for all } j \neq i \}$, and so the array ADJ(i) consists of the set of column indices of the nonzero off-diagonal entries in row i of matrix M. Thus, the problem of representing the matrix M

by a sparse data structure is one of finding a compact way of storing and referencing the n arrays ADJ(1) through ADJ(n). The sparse data structure we use in roadmap was first introduced, implemented, and applied by Gustavson (1973).

Suppose we combine the n arrays ADJ(1) through ADJ(n) to form a single array LADJ defined by

$$LADJ = [\ ADJ(1), ADJ(2), \ \ldots\ , ADJ(n)\ ].$$

Also, let IADJ be an n+1 single array

$$IADJ = [IADJ(1), IADJ(2), \ldots , IADJ(n+1)]$$

with elements defined by

$$IADJ(i) = 1 + \sum_{j=1}^{i-1} DEG(j)\ , \qquad i = 1, \ldots , n+1.$$

Since DEG(i) = |ADJ(i)|, the integers IADJ(i) and IADJ(i+1) -1 are pointers to the first and last vertices of ADJ(i) on array LADJ, respectively, and so using standard Matlab notation (MathWorks, 1990) we obtain the following two equalities:

$$ADJ(i) = LADJ(IADJ(i):IADJ(i+1) - 1)\ ,$$

and

$$DEG(i) = IADJ(i+1) - IADJ(i), \qquad i = 1, \ldots , n.$$

Thus the data structure for the zero-nonzero structure of matrix M is a pair of single arrays (IADJ, LADJ), where IADJ consists of n+1 pointers and LADJ consists of 2|E| column indices. George and Liu (1981) refer to the pair of arrays (IADJ, LADJ) as the adjacency structure pair of matrix M.

The entire roadmap program is given in figures 1 through 8. In writing this program, our main objective was to establish a concise one-to-one correspondence between the implementation and the high-level language used in Kevorkian (1993) so that all theoretical parts of the work can be conveniently traced and studied. All parameters, variables and arrays used in the roadmap are described in detail in Kevorkian (1993).

The procedure "sprsdata" called in roadmap queries the user for the dimension n of matrix M and requires from the user the adjacency structure pair (IADJ, LADJ) of M. Procedure sprsdata also computes the single array DEG and monitors the integrity of the input data by checking the correctness of three relations. These are as follows:

$$\begin{aligned} |IADJ| &= n+1, \\ |LADJ| &= IADJ(|IADJ|) - 1, \\ LADJ(i) &\le n, \qquad \text{for all } i \le |LADJ|. \end{aligned}$$

If any of these three relations is violated, the program halts and prints the message "check data." The program also halts if the following equality holds

$$|LADJ| = n \times (n - 1)$$

since G is a clique in this a case; and so no sparsity will exist in G.

Procedure "search" called in roadmap computes the set of vertices S. If the set S is nonempty, then roadmap calls procedure "dfs" to compute the connected components of the induced subgraph G(V-S). Otherwise, roadmap calls procedure "cliques" to compute independent cliques in the original graph G.

The procedure "classify" called in roadmap categorizes the connected components of G(V-S) into cliques and noncliques using parameters computed in dfs. If a connected component G(U) is a clique, then procedure classify uses results established in Kevorkian (1993) to classify G(U) into one of four distinct types of cliques. If G(U) is not a clique, then procedure classify calls procedure cliques to compute independent cliques in the nonclique connected component G(U).

The clique connected components of induced subgraph G(V-S) combined with the independent cliques computed in each of the nonclique connected components of G(V-S) form the independently solvable smaller tasks that can be executed in parallel on different processors of a parallel architecture computer.

3. BLOCK BORDERED DIAGONAL FORMS USING ROADMAP

Procedure bbdf given in figure 9 uses the program roadmap to generate a permutation matrix P such that PMP^T is a block bordered diagonal matrix satisfying all three properties of the vertex partition $\Pi^* = (V_1, V_2, \ldots, V_r, S^*)$. These properties are briefly stated in procedure roadmap shown in figure 1, and covered in more depth in Kevorkian (1993). By the first two properties of the vertex partition, PMP^T is an (r+1)-by-(r+1) block bordered diagonal matrix such that each of the leading r diagonal blocks is a full matrix. By the third property, every principal submatrix in M that corresponds to an interior clique in the graph of M is a diagonal block in PMP^T. In part 1 of this work (Kevorkian, 1993), we have shown that the symbolic factorization of a matrix corresponding to an interior clique does not produce any fill-in. Block bordered diagonal forms computed by the program roadmap are thus well-suited for sparse block factorizations.

The procedure bbdf consists of three distinct parts. The first part uses the adjacency structure pair (IADJ, LADJ) to generate a Boolean form of matrix M. The second part uses the array QUEUE and properties of arrays SN and VC to compute the vertex ordering σ. Permuting the rows of M in the sequence given by σ produces the block bordered diagonal form of matrix M. The third and last part of procedure bbdf uses the signs of the elements placed on IQUEUE to generate an r+1 array such that the ith element is a pointer to the first row of the ith diagonal block in PMP^T. While modifying the array IQUEUE in procedure bbdf, we make sure that the pointers to the starting vertices of independent cliques computed in cliques retain their original negative signs. This way we are able to use the array TYPE computed

in roadmap to relate each diagonal block in PMP^T to the type of clique it is associated with in G.

The first two parts of procedure bbdf are straightforward and easy to follow. The third part is more complicated, requiring the following property of array IQUEUE for correctness.

Lemma 1. Let S be the set of vertices computed in procedure search. Let i and j be any two consecutive elements on the array IQUEUE at the completion of roadmap. Then the set of vertices on QUEUE(|i|:|j|-1) is a subset of the vertex set S*- S if and only if $i < 0$ and $j > 0$.

Proof. Let i and j be any two consecutive elements on array IQUEUE. Then one of the following two cases must hold.

Case 1. $i > 0$. Then by the construction of procedure dfs, there is a connected component G(U) of G(V-S) such that the vertex v = QUEUE(i) is the starting vertex of the connected component G(U) computed in dfs. Suppose $j > 0$. Then by the last statement in procedure maxclq, it follows that no call was made to procedure cliques in classify at the completion of the connected component G(U). This means that the connected component G(U) is a clique, and so no part of the set of vertices on QUEUE(i:j-1) is in the set S*-S. Now suppose $j < 0$. Since $i > 0$, the integer j must be the first element added to array IQUEUE in procedure maxclq at the completion of connected component G(U). Thus G(U) is not a clique and, furthermore, the vertices on QUEUE(i:|j|-1) comprise the vertex set in the first independent clique computed by procedure cliques in G(U). As a result, no part of the set of vertices on QUEUE(i:|j|-1) can be in the set S*-S. Hence for any $i > 0$ and $j > 0$ or any $i > 0$ and $j < 0$, no part of the set of vertices on QUEUE(i:|j|-1) is in the set S*-S.

Case 2. $i < 0$. Assume $j < 0$. Then by the construction of procedure cliques, the vertex v = QUEUE(|i|) is the starting vertex of an independent clique computed by cliques in some nonclique connected component G(U) of G(V-S), whereas the vertex w = QUEUE(|j|) is either the staring vertex of another independent clique in G(U) or is the starting vertex of a separator computed by cliques in G(U). Therefore for the case where $j < 0$, the vertices on QUEUE(|i|:|j|-1) comprise the vertex set in an independent clique computed by procedure cliques in G(U) and so no part of the set of vertices on QUEUE(|i|:|j|-1) can be in the set S*-S. Finally, suppose $j > 0$. Then v is the starting vertex of the separator computed by cliques in G(U), which means that the set of vertices on QUEUE(|i|:j-1) is in the set S*-S.

This completes the proof.

```
% procedure roadmap
%
%*****************************************************************************************
% Given a sparse symmetric matrix M, roadmap computes in the undirected graph  *
% G = (V, E) of M a vertex partition Π* = (V1, V2, ... , Vr, S*) satisfying the following  *
% three properties:                                                                  *
%    (a) for any two distinct elements Vi and Vj of the partition, no vertex in Vi is  *
%        adjacent to a vertex in Vj,                                                  *
%    (b) every element Vi of the partition induces a clique,                          *
%    (c) interior of every clique in G is an element of the partition.                *
% Program roadmap computes the vertex partition Π* in linear time.                   *
%*****************************************************************************************
%
global IADJ LADJ LEAF NGU RANKE SN TEST U VC
sprsdata                                  % user provided input (n, IADJ and LADJ)
QUEUE = [];                               % store connected components of G(V-S)
IQUEUE = [];                              % pointers to "roots" and "end" of QUEUE
TYPE = [];                                % classify "type" of connected component
LEAF = 0;                                 % pointer to last vertex placed on QUEUE
VC = zeros(1,n);                          % mark all vertices "new"
SN = zeros(1,n);                          % initialize separator numbers to zero
TEST = zeros(1,n);                        % initialize Boolean array TEST to zero
search                                    % compute the set S = { v | SN(v) = 1}
if any(SN == 1)                           % if S is nonempty then
    dfs                                   %     compute connected components
else                                      % else
    ROOT = 1;                             %     pointer to root vertex of V
    LEAF = n;                             %     pointer to end vertex of V
    QUEUE = [ROOT:LEAF];                  %     place entire vertex set V on QUEUE
    IQUEUE = [ROOT];                      %     pointer to root vertex on QUEUE
    TYPE = [0];                           %     G is a regular graph (& not a clique)
    cliques                               %     compute independent cliques of G
end                                       % end
IQUEUE = [IQUEUE, LEAF+1];                % pointer to end of QUEUE
```

Figure 1. Procedure roadmap.

```
% procedure sprsdata
%
%*****************************************************************************************
%   This procedure performs the following four tasks:                                   *
%       (a) queries the user for size of matrix (n);                                    *
%       (b) requires from user the adjacency structure pair (IADJ, LADJ) of matrix;     *
%       (c) tests correctness of input data;                                            *
%       (d) computes the single array DEG (degrees of vertices).                        *
%*****************************************************************************************
%
% query user for size of matrix (n)
   n = input('Enter n (if n <= 1, program quits): ');
   if n <= 1, break, end
   end
% provide adjacency structure pair (IADJ, LADJ)
   if n == 21
       % Illustrative example used in Kevorkian (1993)
       IADJ = [1 6 8 11 14 23 27 32 41 44 47 51 55 61 67 70 75 82 86 90 93 97];
       LADJ = [5 7 8 16 17  13 17  4 14 20  3 13 20  1 6 7 8 14 16 17 18 19];
       LADJ = [LADJ, 5 14 18 19   1 5 8 16 17  1 5 7 11 12 13 16 17 21  10 15 20];
       LADJ = [LADJ, 9 14 15  8 12 13 21  8 11 17 21  2 4 8 11 15 21];
       LADJ = [LADJ, 3 5 6 10 18 19  9 10 13  1 5 7 8 17];
       LADJ = [LADJ, 1 2 5 7 8 12 16  5 6 14 19  5 6 14 18  3 4 9  8 11 12 13];
   end
% test correctness of input data
   RANKI = length(IADJ);
   RANKL = length(LADJ);
   if RANKI ~= n+1
       disp(' IADJ does not have correct length. Check data.');
       break, end
   end
   if RANKL ~= IADJ(RANKI)-1
       disp(' LADJ does not have correct length. Check data.');
       break, end
   end
   if any(LADJ > n)
       disp(' LADJ contains a column index > n. Check data.');
       break, end
   end
   if RANKL == n*(n-1)
       disp(' Off-diagonal part of M is full and so G is a clique.');
       break, end
   end
% compute degrees of vertices
   DEG = IADJ(2:n+1)-IADJ(1:n);
```

Figure 2. Procedure sprsdata.

```
% procedure search
%
%*****************************************************************************************
%    This procedure computes in G = (V, E)  a set of vertices S defined by              *
%                                                                                        *
%        S = { v | there exists in E an edge (v, w) with DEG(v) > DEG(w)} .              *
%                                                                                        *
%    The set S has the property that every interior clique in G is a connected          *
%    component of induced subgraph G(V-S); Corollary 3.1 in Kevorkian (1993).           *
%*****************************************************************************************
%
for v = 1:n-1
    VC(v) = 1;
    for w = LADJ(IADJ(v):IADJ(v+1)-1)
        if VC(w) == 0
            if DEG(v) ~= DEG(w)
                if DEG(v) < DEG(w)
                    SN(w) = 1;
                else
                    SN(v) = 1;
                end
            end
        end
    end
end
VC = zeros(1,n);
```

```
% for each vertex v in V - {n} do
%     mark vertex v "old"
%     for all w adjacent to v do
%         if w is marked "new" then
%             if DEG(v) ≠ DEG(w) then
%                 if DEG(v) < DEG(w) then
%                     w is in S
%                 else
%                     v is in S
%                 end
%             end
%         end
%     end
% end
% mark all vertices "new"
```

Figure 3. Procedure search.

```
% procedure dfs
%
%*****************************************************************************************
%    This procedure computes all connected components of induced subgraph        *
%    G(V-S). We use G(U) to denote a connected component computed in dfs.        *
%*****************************************************************************************
%
for v = 1:n
    if SN(v) == 0
        if VC(v) == 0
            VC(v) = 1
            QUEUE = [QUEUE,v];
            LEAF = LEAF+1;
            ROOT = LEAF;
            IQUEUE = [IQUEUE, ROOT];
            RANKE = 0;
            NGU = [ ];
            cmponent(v)
            RANKU = LEAF-ROOT+1;
            RANKN = length(NGU);
            TEST(NGU) = zeros(1,RANKN);
            classify
        end
    end
end
```

```
% for all v in V do
%     if v is in V-S then
%         if v is marked "new" then
%             mark v "old"
%             v is the root vertex of G(U)
%             pointer to end vertex
%             pointer to root vertex
%             add root pointer to IQUEUE
%             count edges visited in dfs
%             neighborhood of U in G
%             compute G(U)
%             compute size of U
%             compute size of NGU
%             update Boolean array TEST
%             identify "type" of component
%         end
%     end
% end
```

Figure 4. Procedure dfs.

```
% function cmponent(v)

%*****************************************************************************************
%    This function computes neighborhood NGU of U while computing G(U)          *
%*****************************************************************************************
%
function cmponent(v)                          % declare cmponent(v) as function file
for w = LADJ(IADJ(v):IADJ(v+1)-1)             % for all w adjacent to v do
    if SN(w) == 0                             %     if w is in V-S then
        RANKE = RANKE+1;                      %         account for visited edge (v,w)
        if VC(w) == 0                         %         if w is marked "new" then
            VC(w) = 1;                        %             mark w "old"
            QUEUE = [QUEUE, w] ;              %             add w to QUEUE
            LEAF = LEAF+1;                    %             update pointer to end vertex
            cmponent(w)                       %             call cmponent(w)
        end                                   %         end
    else                                      %     else
        if TEST(w) == 0                       %         if w is not on NGU then
            NGU = [NGU, w];                   %             add w to NGU
            TEST(w) = 1;                      %             mark w as vertex on NGU
        end                                   %         end
    end                                       %     end
end                                           % end
```

Figure 5. Function cmponent.

```
% procedure classify
%
%*****************************************************************************************
% This procedure categorizes the connected components of G(V-S) into cliques      *
% and noncliques using the algebraic relation RANKE = RANKU*(RANKU-1).            *
% Subsequently, all clique connected components in G(V-S) are classified into     *
% the four types of cliques C1 through C4 using Corollaries 4.2 and 4.3 in        *
% Kevorkian (1993).                                                               *
%*****************************************************************************************
%
if RANKE == RANKU*(RANKU-1)
    if RANKN == 1
        TYPE = [TYPE, 1];
    else
        R = RANKN+RANKU-1;
        if any(DEG(QUEUE(ROOT:LEAF)) ~= R)
           TYPE = [TYPE, 3];
        else
            TYPE = [TYPE, -2];
        end
    end
else
    TYPE = [TYPE, 0];
    VC(QUEUE(ROOT:LEAF))=zeros(1,RANKU);
    cliques
end
```

```
% if G(U) is a clique then
%     if Cor. 4.2 holds then
%         G(U) is an interior clique
%     else
%         compute integer R
%         if Cor. 4.3 does not hold
%             G(U) is not an si clique
%         else
%             G(U) is an si clique
%         end
%     end
% else
%     G(U) is not a clique
%     mark vertices in G(U) "new"
%     compute ind. cliques in G(U)
% end
```

Figure 6. Procedure classify.

```
% procedure cliques
%
%*****************************************************************************************
% This procedure computes in G(U) independent cliques G(U1), G(U2) , ... , and          *
% their neighborhoods N(U1),  N(U2) , ... ,  in G(U) such that                           *
%      G(U1) is maximal in G(U),                                                         *
%      G(U2) is maximal in G(U - U1 - N(U1)),                                            *
%      G(U3) is maximal in G(U - U1 - U2 - N(U1) - N(U2)),                               *
% and so forth. The sets U1, U2, ... , are placed on array CLQS in that order,           *
% while all neighborhoods are placed on array NBRS. At the completion of                 *
% cliques the vertex set U on QUEUE is replaced by the array [CLQS,NBRS].                *
%*****************************************************************************************
%
CLQS = [];                                  % stores independent cliques
NBRS = [];                                  % stores neighborhoods of ind. cliques
CLQROOT = ROOT;                             % pointers to ind. cliques on CLQS
TAIL = 0;                                   % pointer to last vertex on CLQS
for v = QUEUE(ROOT:LEAF)                    % for each vertex v in G(U) do
    if VC(v) == 0                           %      if v is marked "new" then
        VC(v) = 1;                          %           mark v "old"
        ADJCNT = [];                        %           adj(v) in G(U-CLQS-NBRS)
        maxclq                              %           compute a maximal clique
    end                                     %      end
end                                         % end
QUEUE(ROOT:LEAF) = [CLQS, NBRS];            % replace U on QUEUE by [CLQS,NBRS]
```

Figure 7. Procedure cliques.

```
% procedure maxclq
%
%*****************************************************************************************
%   This procedure computes in induced subgraph G(U-CLQS-NBRS) a maximal      *
%   clique G(C) with starting vertex v (occasionally called the root vertex).  *
%*****************************************************************************************
%
for w = LADJ(IADJ(v):IADJ(v+1)-1)
    if SN(w) == 0
        if VC(w) == 0
            ADJCNT = [ADJCNT, w];
        else
            NBRS = [NBRS, v];
            SN(v) = 1;
            return
        end
    end
end
CLQS = [CLQS, v];
TEST(v) = 1;
RANKC = 1;
for u = ADJCNT
    VC(u) = 1;
    w = LADJ(IADJ(u):IADJ(u+1)-1);
    COUNT = length(find(TEST(w) == 1));
    if COUNT == RANKC
        CLQS = [CLQS, u];
        TEST(u) = 1;
        RANKC = RANKC+1;
    else
        NBRS = [NBRS, u];
        SN(u) = 1;
    end
end
HEAD = TAIL+1;
TAIL = TAIL+RANKC;
TEST(CLQS(HEAD:TAIL))=zeros(1,RANKC);
CLQROOT = CLQROOT+RANKC;
IQUEUE = [IQUEUE, -CLQROOT];
```

```
% for all w adjacent to v do
%     if w is on U but not on NBRS then
%         if w is marked "new" then
%             add w to ADJCNT
%         else
%             reject v as starting vertex
%             avoid duplicates of v
%             return to cliques
%         end
%     end
% end
% C = [v]; (v is starting vertex of G(C))
% set TEST(v) = 1
% RANKC = |C|
% for each vertex u on ADJCNT do
%     mark u "old"
%     w = ADJ(u)
%     COUNT = |ADJ(u) ∩ C|
%     if u is adjacent to all w in C then
%         C = [C, u]
%         set TEST(u) = 1
%         update size of C
%     else
%         u is in neighborhood of C
%         avoid duplicates of u
%     end
% end
% pointer to C(1) on CLQS
% pointer to C(|C|) on CLQS
% reset TEST to zero
% pointer to next starting vertex
% add pointer (negated) to IQUEUE
```

Figure 8. Procedure maxclq.

```
% procedure bbdf
%
%*****************************************************************************************
% This procedure uses the output of program roadmap to generate a permutation  *
% matrix P such that PMP^T is a block bordered diagonal matrix satisfying all three  *
% properties of vertex partition Π* = (V1, V2, ... , Vr, S*).                        *
%*****************************************************************************************
%
M = zeros(n,n);
for v = 1:n
    M(v,v) = 1;
    w = LADJ(IADJ(v):IADJ(v+1)-1);
    M(v,w) = 1;
end
%
S = QUEUE(find(SN(QUEUE) == 1);
QUEUE = QUEUE(find(SN(QUEUE) ==0));
QUEUE = [QUEUE,S, find(SN==1 & VC==0)];
M = M(QUEUE, QUEUE);
%
if length(S) > 0
    SUM = 0;
    k = 2;
    i = IQUEUE(2);
    for j = IQUEUE(3:length(IQUEUE))
        if i < 0 & j > 0
            SUM = SUM+i+j;
        else
            IQUEUE(k) = i-sign(i)*SUM;
            k = k+1;
        end
        i = j;
    end
end
```

```
% use (IADJ, LADJ) to compute M
% create an n by n zero matrix M
% for all v in V do
%     set M(v,v) = 1
%     w = vertices adjacent to v
%     set M(v,w) = 1
% end
% use array QUEUE to compute P
% compute bordered part on QUEUE
% compute block diagonal part
% vertex ordering σ (placed on QUEUE)
% block bordered diagonal form of M
% modify array IQUEUE
% if |S| > 0 then
%     sum sizes of bordered parts
%     pointer for elements on IQUEUE
%     second element  on IQUEUE
%     for j = IQUEUE(3:|IQUEUE|) do
%         if i < 0 and j > 0 then
%             update SUM
%         else
%             update IQUEUE
%             increment k
%         end
%         i = jth element on IQUEUE
%     end
% end
```

Figure 9. Procedure bbdf.

4. FUTURE WORKS

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

Also, we are currently working on an extension of roadmap (Kevorkian, in preparation-a) that exploits parallelism in the original problem as well as subsequent Schur complements until no further parallelism remains to exploit. Such a recursive version of roadmap will be ideally suited for problems in which the original matrix and the Schur complement of the pivot block selected by roadmap are very large sparse matrices. Large sparse problems with large Schur complements are frequently encountered in linear and nonlinear programming optimization problems (Kevorkian, 1993).

5. CONCLUSION

We have presented a detailed computer implementation of a linear-time parallelization tool that automatically decomposes a large arbitrary sparse symmetric system of equations into independently solvable smaller tasks for execution on different processors of a parallel architecture computer.

6. REFERENCES

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

Gustavson, F. G. 1973. "Some basic techniques for solving sparse systems of linear equations," in *Sparse Matrices and Their Applications*, D. Rose and R. Willoughby, Eds., Plenum Press, NY, pp. 41-52.

Kevorkian, A. K. 1993 (Mar). "Decomposition of large sparse symmetric systems for parallel computation. Part 1. Theoretical Foundations," NCCOSC/NRaD Technical Report TR1572.

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.

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
March 1993

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
DECOMPOSITION OF LARGE SPARSE SYMMETRIC SYSTEMS FOR PARALLEL COMPUTATION
Part 2: Parallelization Tool Roadmap

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
TR 1601

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; distributikon is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this report we give a complete computer implementation of an automated algorithmic tool for exploiting the parallelism hidden in the sparsity structure of large symmetric matrices with regular and irregular structures. With this parallelization tool, large sparse symmetric systems of equations are automatically decomposed into independently solvable smaller tasks that can be executed in parallel on different processors of a parallel architecture computer.

14. SUBJECT TERMS
cliques
separators
vertex partition
fill-in
simplicial vertices
parallel computation
symbolic factorization

15. NUMBER OF PAGES
23

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
SAME AS REPORT