Purdue University

Purdue e-Pubs

Department of Computer Science Technical Reports | Department of Computer Science

1990

PARALLEL SPARSE: Data Structure and Organization

Mo Mu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
90-974

Mu, Mo and Rice, John R., "PARALLEL SPARSE: Data Structure and Organization" (1990). *Department of Computer Science Technical Reports.* Paper 827.
https://docs.lib.purdue.edu/cstech/827

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries.
Please contact epubs@purdue.edu for additional information.

PARALLEL SPARSE:
DATA STRUCTURE AND ORGANIZATION

Mo Mu
John R. Rice

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #974
May 1990

PARALLEL SPARSE:
Data Structure and Organization

Mo Mu*
and
J.R. Rice**

Computer Sciences Department
Purdue University
Technical Report CSD-TR-974
CAPO Report CER-90-17
May, 1990

* Work supported in part by National Science Foundation grant CCR-8619817.
** Work supported in part by the Air Force Office of Scientific Research grant, 88-0243, and the Strategic Defense Initiative through Army Research Office contract DAAL03-86-K-0106.

PARALLEL SPARSE:
Data Structure and Organization

Mo Mu * and John R. Rice**

Computer Science Department
Purdue University
West Lafayette, IN 47907
Technical Report CSD-TR-974
CAPO Report CER-90-17
May 1990

ABSTRACT

PARALLEL SPARSE is an algorithm for the direct solution of general sparse linear systems using Gauss elimination. It is designed for distributed memory machines and has been implemented on the NCUBE-7, a hypercube machine with 128 processors. The algorithm is intended to be particularly efficient for linear systems arising from solving partial differential equations using domain decomposition with a nested dissection ordering. PARALLEL SPARSE is part of the Parallel ELLPACK system.

This report assumes the reader is familiar with the general approach of parallel sparse and it provides detailed information on three aspects of PARALLEL SPARSE:

1. The data structures used to represent the matrix, the modifications in eliminating unknowns and the dependencies between processors.
2. The data structures that relate the assignment of actual hypercube processors to computational processes.
3. The organization of the codes that run on the hypercube host and on the hypercube nodes.

Dynamic data structures are used unlike most other sparse matrix codes. These are more complex but provide better flexibility to handle PDE problems. Much of the complexity seen here compared to traditional other codes is due to the fact that we handle general matrices instead of only symmetric ones.

* Work supported in part by National Science Foundation grant CCR-8619817.
** Work supported in part by the Air Force Office of Scientific Research grant, 88-0243 and the Strategic Defense Initiative Office contract DAAL03-86-K-0106.

I. INTRODUCTION

PARALLEL SPARSE is an algorithm for the direct solution of general sparse linear systems using Gauss elimination. It is designed for distributed memory machines and has been implemented on the NCUBE-7, a hypercube machine with 128 processors. The algorithm is intended to be particularly efficient for linear systems arising from solving partial differential equations using domain decomposition with a nested dissection ordering. PARALLEL SPARSE is part of our Parallel Ellpack system [Houstis, Rice, 1989].

There have been a lot of work on developing parallel algorithms for solving sparse systems using Gauss elimination (e.g., [George, Heath, Liu, Ng, 1988], [Mu, Rice, 1989], [Duff, 1986]). Elimination tree is a useful tool in this field. This concept is first introduced from the algebraic point of view of the sparse structure of matrices. For a symmetric matrix A, one determines the sparse structure of its Cholesky factor L with $A = LL^T$ by applying symbolic factorization to A and then defines the elimination tree of A from the structure of L with each tree node corresponding to one unknown. This tree reflects the dependency of unknowns during elimination and therefore can be used to exploit the parallelisms inherent in it. If A is nonsymmetric, the definition of the elimination tree is not as natural. It uses traditional elimination trees by doing either a symbolic Cholesky factorization on A^TA (or $A + A^T$) for the coefficient matrix A [George, Liu, Ng, 1988] or a modified symbolic LU factorization with ''worse case'' assumptions in the fill [George and Ng, 1988]. On the other hand, tree shapes affect their potential parallelism. A good elimination tree should be balanced, wide and short. One can improve tree shapes by reordering unknowns/equations [Liu, 1988].

A geometric (or physical) approach to develop parallel sparse solvers for solving PDE problems is suggested in [Mu and Rice, 1989], [Mu and Rice, 1990a], which has several advantages over some standard algebraic approaches. By extending the conventional elimination tree to a block one, it naturally allows non-symmetry in the linear system, avoids symbolic factorization, leads to a well shaped (block) elimination tree, and also allows people to flexibly combine various ideas in different regions according to the local information of geometry and physics of PDEs for assignment, indexing and algebraic solution. These ideas are developed more systematically in [Mu, Rice, 1990b].

II. ALGORITHM DESCRIPTION

II.a. Block Elimination Tree

The algorithm for PARALLEL SPARSE is based on the fact that the unknown dependency during elimination can be expressed by a (binary) **block elimination tree** as shown in Figure 2.1. Each node corresponds to a block of unknowns/equations. The indexing of unknowns is bottom level to top level by blocks, i.e., if unknowns u and v are in nodes S and T, respectively, and S is on a higher level than T is, then the index of u is larger than that of v. The indexing of blocks on the same level and the local indexing within each node is essentially arbitrary.

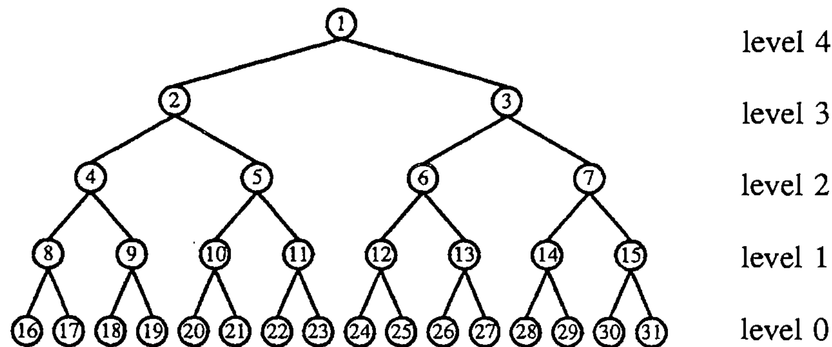


Figure 2.1. Block elimination tree with five levels

We can always get such a block elimination tree by our geometric approach to solving PDEs. For simplicity of exposition, we consider a PDE problem on a rectangular domain Ω, the approach can be extended easily to general domains. Suppose we have $p\,(= 2^{2d})$ processors available, 2^d in each direction. By domain decomposition Ω is divided into p subdomains Ω_{ij}, i, $j = 1, 2, ..., p^{1/2}$ as shown in Figure 2.2. One puts a local grid on each subdomain Ω_{ij} and discretizes the local problem whose solution U_{ij} only depends on unknowns at grid points of $\partial\Omega_{ij}$, the boundary of Ω_{ij}.

Initially, all interior unknowns U_{ij} are eliminated locally as in the standard domain decomposition approach. This step is obviously totally parallel. Then, all processors participate in eliminating interface unknowns. To exploit more parallelism, we use dissection in alternating directions to partition the interface set into several levels suitable for a hypercube machine, each level consists of several **separators**, groups of unknowns which separate regions. The partition, which we call the **one way nested dissection decomposition**, is shown by Figure 2.3 with circles representing the unknowns interior to the subdomain Ω_{ij}, the boxes representing the **separators**. For simplicity, they are all called **subdomains** of this domain decomposition in the nested

dissection manner and are numbered from top level to bottom level as shown in Figure 2.3.

Ω_{11}	Ω_{12}	Ω_{13}	Ω_{14}
Ω_{21}	Ω_{22}	Ω_{23}	Ω_{24}
Ω_{31}	Ω_{32}	Ω_{33}	Ω_{34}
Ω_{41}	Ω_{42}	Ω_{43}	Ω_{44}

Figure 2.2. Domain decomposition of a rectangle for $d = 2$.

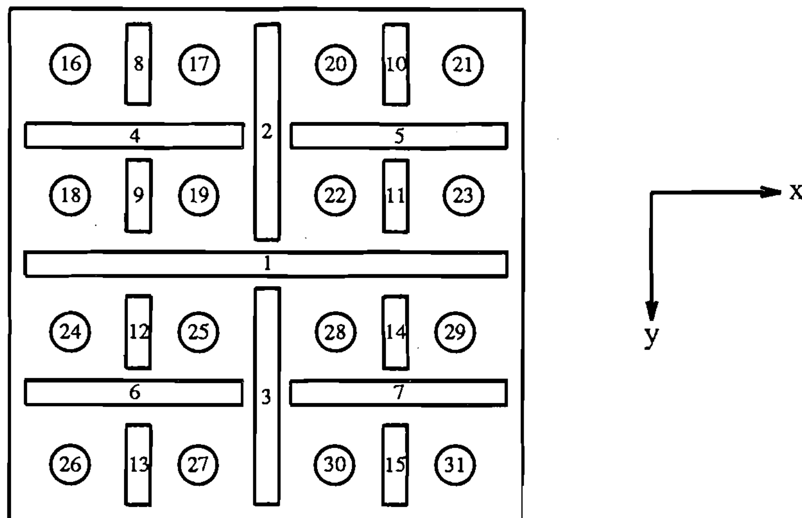


Figure 2.3. Partition of the subdomain interfaces in Figure 2.2 using one way nested dissection. The circles (16–31) represent the 16 groups of interior unknowns and the boxes represent groups of interface unknowns or separators. All boxes of the same size are on the same level of the elimination tree.

This domain decomposition naturally inherits certain parallelism because the PDE discretization process leads to a local or boundary dependence property for interfaces. For example, if we consider the union of subdomains 16, 17, 8 as a more general subdomain Ω'_8 then the local interior solution set U'_8 is uniquely determined by unknowns on $\partial\Omega'_8$. This relation holds similarly for groups at higher levels of the dissection decomposition for the unknowns arising in PDE applications. This local dependency can thus be described by a binary tree as shown in Figure 2.1 with each tree node corresponding to a subdomain in the decomposition as shown in Figure 2.3.

Block elimination trees play a role similar to that the standard **elimination tree** plays in exploiting certain parallelism. However, each node corresponds now to a block of unknowns/equations rather than a single unknown/equation. In other words, we are seeking the parallelism in the block sense no matter what local properties the linear system has within each node locally. This block elimination tree has the following properties: (a) Each node corresponds to a set of unknowns from one location, local ordering and lack of symmetry in the linear system do not affect the tree structure, (b) Eliminating the unknowns in a node only has effects on its ancestors, (c) The elimination of nodes that are not descendants/ancestors of one another are independent of one another.

II.b. Parallelism

There are four kinds of potential parallelism here. First, elimination steps in independent nodes of the block elimination tree can execute simultaneously. To see this, consider two independent nodes S and T. Property (b) says that eliminating S and T will not affect with each other, while Property (c) means that the effects on their common ancestors, if any, are also independent. Therefore, we can independently start to eliminate a node as soon as all of its descendants have been eliminated. We call this the **outer parallelism**. Second, if there are several processors available for a single node, we can also exploit **inner parallelism** within the node. This does not occur at leaf nodes if each leaf node has only one processor as in usual cases even though it represents a sparse subproblem. For the other nodes we apply various efficient parallel dense solvers to exploit the inner parallelism. Third, the tasks to modify an equation to eliminate an unknown (or simply, a *modification*) are independent for different equations, just as for dense matrices. Finally and fourth, modifications, even on the same equation, due to independent descendant nodes can be performed in arbitrary order and hence in parallel. This parallelism cannot be fully exploited for distributed memory machines because one usually assigns each equation (and therefore the associated modification task on it) to a single processor. But, together with the first type of parallelism, it is related to the pipelining technique in the sense that one can start the

parallelism, it is related to the pipelining technique in the sense that one can start the elimination in a node S without waiting for the completion of elimination in other nodes independent of S, even though the associated modifications have been ready for manipulation. For more details, see [Mu, Rice, 1990b].

An algorithmic description of a (pipelining) distributed parallel sparse algorithm for a processor P, as in the module **PARALLEL SPARSE**, is as follows.

```
for level l from bottom to top, do:
    for each node S, with equations assigned to P, on level l, do:
        if l=0, then
            elim_local(S)
        else
            elim_global(S)
            elim_local(S)
        endif
    end of S loop
end of l loop
```

The **elim_local**(S) procedure has processor P participating in eliminating unknowns in S by performing the associated modifications on equations assigned to P. For those equations of S assigned to processor P, it also has to calculate the corresponding multiplier vectors and to send them to other processors. When level l=0, one usually assigns S to only one processor and on this node it is therefore a sequential sparse solver. Otherwise, it is a sort of parallel dense solver using processors assigned to S. The **elim_global**(S) procedure has processor P performing the modifications on its equations due to eliminating unknowns in the descendants of S in the block elimination tree which have no equations assigned to P. Therefore, the effects of elimination at these nodes have not yet been processed by P.

II.c. Assignment

By assigning an unknown to a processor we mean assigning both the problem data and the factorization subtask associated with this unknown. To achieve high parallelism, load balancing and low communication costs we want to (a) avoid assigning independent nodes to the same processor, and (b) assign processors to a single node so as to have minimal communication connections. In this report, we assume a **subtree-subcube** type assignment for these purposes. It is a top to bottom process. Assume that the number of processors used is equal to that of nodes on the leaf level 0. First, the root node of the elimination tree is assigned to the whole hypercube and then the

hypercube is split into two subcubes to which the two descendent subtrees are assigned. This process goes on recursively until all subtrees become assigned to single processors. The assignment within each node is potentially arbitrary. For various subtree-subcube assignments, see [George, Liu, Ng, 1987] and [Mu, Rice, 1990a].

For a *subtree–subcube* type assignment, the general sparse algorithm as described above can be simplified. For each processor P there is an *elimination path* in the block elimination tree from bottom to top with exactly one assigned node S_l on each level l for $l = 0, 1, ..., L$. Therefore, the algorithm can be rewritten as follows.

```
elim_local(S_0)
for l = 1 to L, do:
     elim_global(S_l)
     elim_local(S_l)
end of l loop
```

III. DATA STRUCTURES FOR NODE PROCESSORS

This section describes the data structure suitable for sparse matrix computations on a distributed memory machine as developed for the Parallel ELLPACK system and its module PARALLEL SPARSE.

Suppose processor P is assigned three unknowns $u1$, $u5$ and $u8$ and equations 1, 5 and 8 are stored at P. For performing the operations of Gauss elimination in P, the processor P needs to know the nonzero structure of the matrix A (or LU) during the elimination process. We illustrate the data structures used with the example in Figuare 3.1 which shows the nonzero structure of A as it is known to P.

```
          1 2 3 4 5 6 7 8 9 10
          X 0 X 0 0 0 X 0 X 0    equation 1
          X · · · X · · X · ·
          0 · · · 0 · · X · ·
          0 · · · Z · · Z · ·
    A  =  0 0 X X X 0 Z X Z 0    equation 5
          X · · · 0 · · 0 · ·
          0 · · · 0 · · X · ·
          0 0 0 X Z Z Y X Y X    equation 8
          0 · · · 0 · · 0 · ·
          X · · · Z · · Z · ·
```

Figure 3.1. Example matrix used to illustrate the data structure.

The following notation is used in Figure 3.1.

X = original nonzero entry in the matrix A

Y = nonzero created by processing of equations and unknowns assigned to the processor P

Z = nonzero created by processing of equations and unknowns assigned to other processors.

Two important characteristics in our geometric approach are that no symbolic factorization is used and symmetry is not assumed. We therefore use a dynamic data structure instead of a static one as used with symbolic factorization. Three data structures – A-INFO, M-INFO and C-INFO – are used to represent the information about the sparse matrix during the elimination. Each of these is dynamically updated, A-INFO contains both numeric and symbolic data, the others contain only symbolic data (indices and pointers). The A-INFO structure only represents the information of A's rows in processor P, while the M- and C-INFO structures represent the necessary information about A's columns due to a lack of symmetry.

III.a. A-INFO: Data Structure for the Distributed Sparse Matrix. a, id_col, len_a, hdr_rowl, hdr_rowu, ptr_a

This data structure encodes the information of the values and locations of the matrix entries. It is dynamically updated as new nonzero entries are created during the elimination process. It is equation (or row) oriented information.

```
              1 2 3 4 5 6 7 8 9 10
hdr_rowl(1)→  X 0 X 0 0 0 X 0 X 0   hdr_rowu(1)
              X · · · X · · X · ·
              0 · · · 0 · · X · ·
              0 · · · Z · · Z · ·
hdr_rowl(2)   0 0 X X X 0 Z X Z 0   hdr_rowu(2)
              X · · · 0 · · 0 · ·
              0 · · · 0 · · X · ·
hdr_rowl(3)   0 0 0 X Z Z Y X Y X   hdr_rowu(3)
              0 · · · 0 · · 0 · ·
              X · · · Z · · Z · ·
```

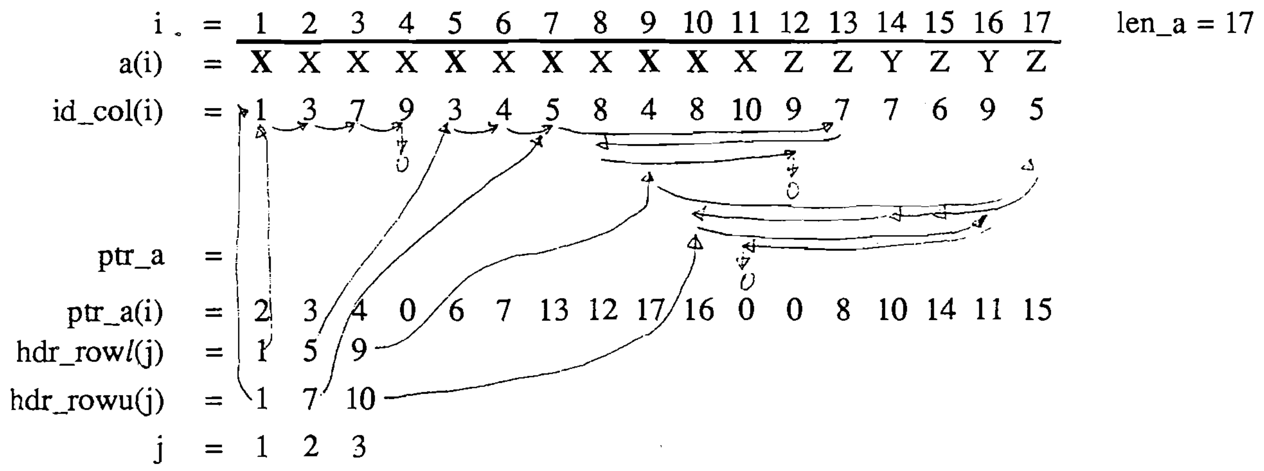


Figure 3.2. The A-INFO structure for the example in Figure 3.1.

III.b. M-INFO: Data Structure for Modification Information. idg_m, len_m, hdr_m, ptr_m, hdrad_m

This data structure encodes the information of locations of matrix entries which must be converted to zeros at the corresponding elimination step. In other words, modifications by the pivot row are needed on the rows with these entries. It is basically unknown (or column) oriented.

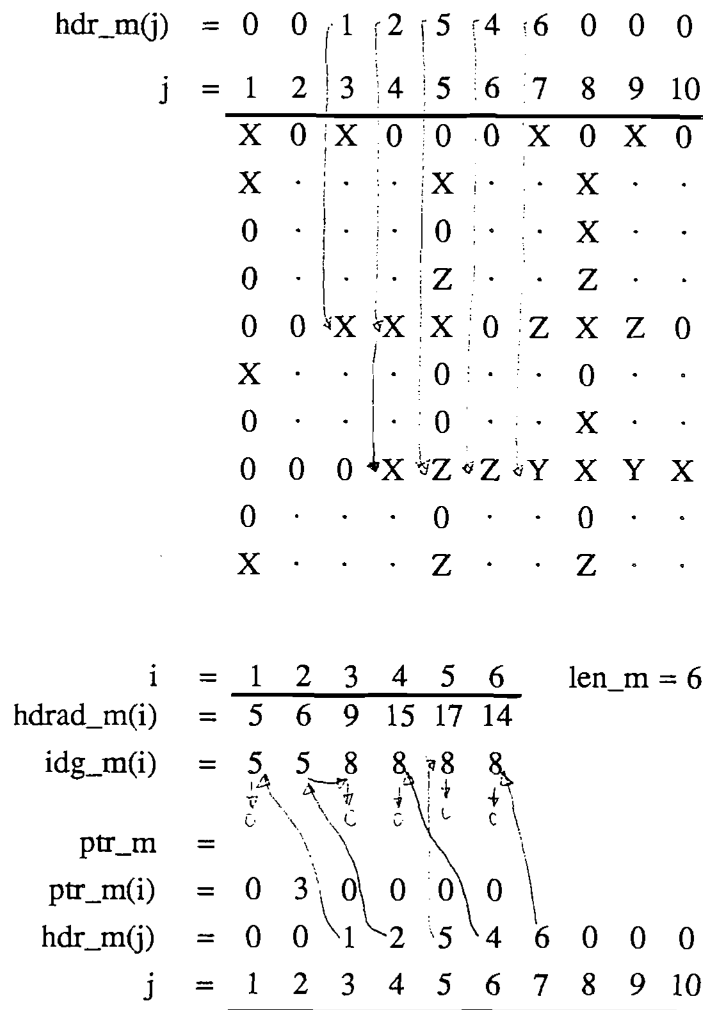


Figure 3.3. The M-INFO structure for the example in Figure 3.1.

```
X 0 X 0 0 0 X 0 X 0
X · · · X · · X · ·
0 · · · 0 · · X · ·
0 · · · Z · · Z · ·
0 0 X X X 0 Z X Z 0
X · · · 0 · · 0 · ·
0 · · · 0 · · X · ·    ptr = hdr_m(6) = 4
0 0 0 X Z Z Y X Y X
0 · · · 0 · · 0 · ·
X · · · Z · · Z · ·
```

idg_m(ptr) = 8 (row index of "**Z**")

hdrad_m(ptr) = 15 (address of "**Z**" in array A, i.e., a(15) = "**Z**")

Figure 3.4. The example of Figure 3.1 showing the structure variables for the Z shown in bold.

III.c. C-INFO: Data Structure for Communication Information. idg_c, len_c, hdr_c, ptr_c

This data structure encodes information about what matrix information is needed from other processors (that is, from equations not assigned to the processor *P*). It is used (along with the matrix data structure) also to determine where new nonzero entries will be created during the eliminations.

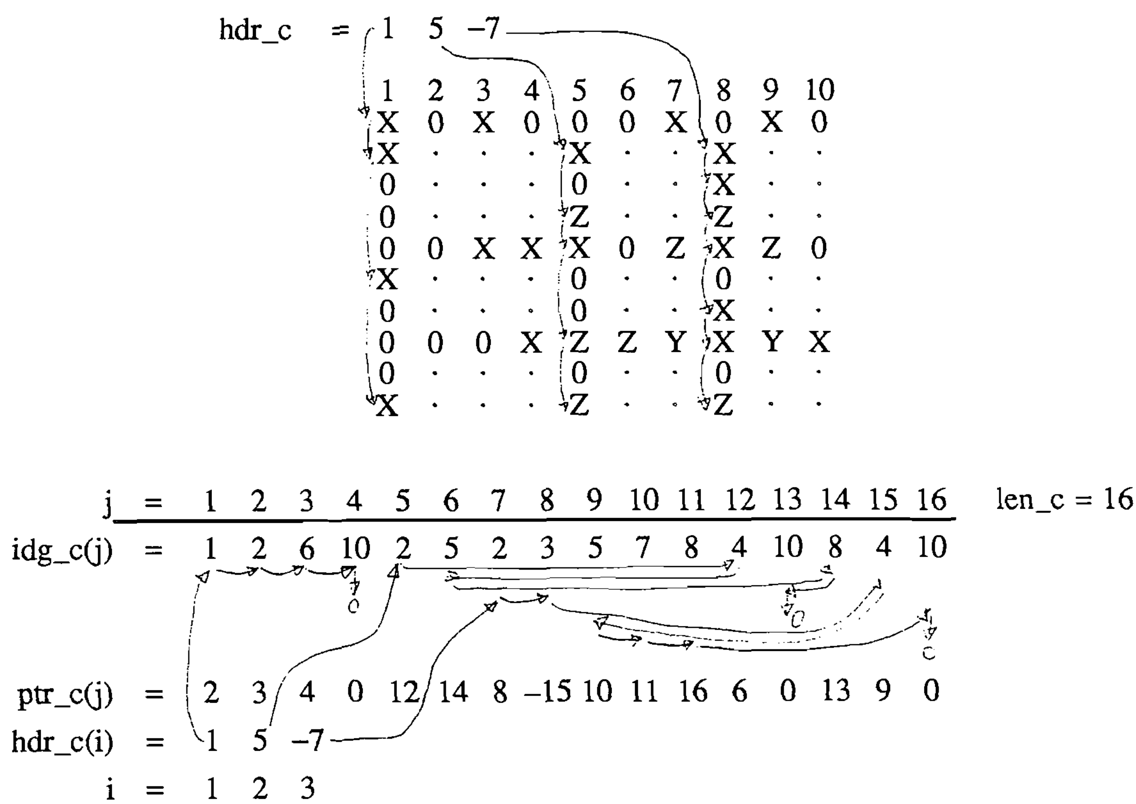


Figure 3.5. The C-INFO structure for the example of Figure 3.1.

The value for hdr_c or ptr_c is negative when the processor that holds a particular equation number has already passed on its information about where it might create fill-in and this information has been processed by the processor assigned the unknown associated with the column considered.

IV. DATA STRUCTURES FOR THE GLOBAL COMPUTATION

This section describes the data structures for the global computations in the module PARALLEL SPARSE as developed for the Parallel ELLPACK system and which uses the subtree-subcube method to assign processor to nodes of the elimination tree as described in Section II. These structures are used in conjunction with the dynamic data structures for a sparse matrix as described in Section III.

We describe in order the following data structures used in the global computation.

1. The elimination tree of the linear system:
   id1_nd, idl_nd
2. The assignment of equations to processors:
   cidg_eqn, ng_eqn, id_leaf
3. The relation between nodes of the elimination tree:
   lr
4. The pivot communication among processors:
   rbuf, len_rbuf
5. The buffer for the C-Info data structure communicated among processors:
   ibuf, len_ibuf, n_rows
6. The auxiliary information used in forming ibuf:
   hdr_proc, ptr_id

These data structures are discussed below, the context and notation of this discussion are as follows:

- The elimination tree is a complete binary tree
  S, T denote generic nodes of the tree (not their index)
- Each node S consists of equations in consecutive order
  i1, i1+1, ..., ilast

  $i1 = 0$ means that S is empty, which handles those applications where the elimination trees are not really complete binary trees.
- The nodes are numbered top-to-bottom and left-to-right.
- The number of processors equals the number of leaf nodes in the elimination tree and there is a one-to-one correspondence between them.

- P, Q, R denote generic processors

IV.a. The Elimination Tree

If the node S is the n-th node then

$$\text{id1_nd(n)} = \text{i1} \qquad \text{idl_nd(n)} = \text{ilast}$$

IV.b. The Assignment of Equations to Processors

Let *ng_eqn* be the total (global) number of equations (unknowns) in the linear system. Then, for processor P, we have the array *cidg_eqn* to identify and locate the equations assigned to P defined, for i = 1,2,..., *ng_eqn* as follows:

$$cidg_eqn\,(i) = k, \text{ or } -k$$

if the i-th equation of the global system is the k-th equation assigned to P, or is assigned to processor Q with $id = k$. Processor P is assigned exactly one leaf node, say the m-th, and we have

$$id_leaf = \text{m}$$

Note that P may have other equations and nodes assigned to it from higher up in the tree. Thus the Gauss elimination for the equations assigned to P starts at the leaf node (*id_leaf*) and proceeds on a path up toward the root node of the elimination tree.

IV.c. The Relation between Nodes of the Elimination Tree

Let S and T be nodes in the elimination tree with, for concretness, T a descendent of S. They are called *related* if and only if elimination in node T affects equations in node S or the ancestors of S. If T is a son of S, then S and T are related.

For each node S in the elimination path of a leaf node S' we identify its son Son_S which is not in the elimination path. Starting at the leaves of the tree and skipping the first such node, we then list all the nodes T's in the subtree STR_s rooted at node Son_S excluding Son_S itself. For each T in STR_s, we use a logical value $l(= l_{T,S})$ to indicate the relation between T and S:

$$\ell = \begin{cases} \textit{true} & \text{if } T \text{ and } S \text{ are related} \\ \textit{false} & \text{otherwise} \end{cases}$$

If the depth of the elimination tree is d (so there are $2^d - 1$ nodes in the tree) then there are $2 + 6 + 14 + \cdots + (2^i - 2) + \cdots = \sum_{i=2}^{d} (2^i - 2) = 2^{d+1} - 2d - 2$ such pairs of (S, T), where there are $(d-1)S$'s corresponding to levels i from 2 to d, and from each S on level i, there are $(2^i - 2)T$'s in STR_s. Therefore, we allocate an array lr of length $2^{d+1} - 2d - 2$ to store these logical values l. The structure of the array lr is illustrated by the example below with $d = 4$ and S' the 24th node. Then the elimination path of S' is 24, 12, 6, 3, 1 and the sons Son_S are 13, 7 and 2 (with 25 omitted). The whole list of nodes in the elimination is structured as follows:

level i	2	3	4
id of S	6	3	1
id of Son_S	13	7	2
nodes in STR_s j =	26,27 1, 2,	28,29,30,31,14,15 3	16,17,18,19,20,21,22,23,8,9,10,11,4,5 ... , 22

The true-false value of lr is determined during the computation.

The Gauss elimination proceeds in the order described above, i.e., each processor follows the elimination path (outer loop runs over S). For each son Son_S of a node S in this path, the algorithm of elim_global(S) checks each node T in STR_s (inner loop runs over T) using the lr values to see if eliminating unknowns in T causes modifications in processor P. The counter j goes through these checks in the proper order and the $lr(j)$ variable indicates if any modification action is needed, i.e., if T and S are related. Note that the nodes in STR_s need not necessarily be related to S, but the potential exists, i.e., not all the values in $lr(j)$ are true.

IV.d. The Pivot Row Communication Among Processors

Assume that the i-th row (equation) is assigned to processor P and, when it is ready to be used as a pivot (all entries before the i-th one are zero), it has k non-zero elements (including the ith one = the pivot). The array *rbuf* has the values of non-zero elements and their column indices (2k entries in total) in the order: value, index, value, index, ...

Thus, for example, if the i-th row appears at the i-th elimination step as

index	:				i	i+1		i+3			i+6	i+7	i+8	
value	:	0	...	0	x	x	0	y	0	0	x	z	x	0

then

$$\text{len_rbuf} = 12$$

$$\text{rbuf} = x, i, x, i+1, y, i+3, x, i+6, z, i+7, x, i+8$$

IV.e. The Buffer for the C-INFO Data Structure Communicated Among Processors

Assume the i-th row (equation) is assigned to P and the i-th unknown is about to be eliminated. Processor P knows the non-zero structure of both the i-th row and i-th column of A at that time. The non-zero elements in the i-th row beyond the pivot must be added in the corresponding positions of all equations which have non-zero entries in the i-th column. A simple example is given below.

Processor	P	Q		P			Q	R	Q	
column	i	i+1		i+3			i+6	i+7	i+8	
row = i	x	x	0	y	0	0	x	z	x	0
i+1	x									
i+2	y									
	0									
i+4	x									
	0									
	0									
	0									
	0									
i+9	z									

Here unknowns i+1, i+3, i+6, i+7 and i+8 might create fill-in (non-zero coefficients) in equations i+1, i+2, i+4 and i+9. The C-INFO structure holds information which must

be updated if this happens. When processor Q eliminates the (i+1)-st unknown (or (i+6)-th or (i+8)-th), it must first obtain information from processor *P* about what happened in equations i+1, i+2, i+4 and i+9 during the elimination of the *i*-th unknown. Similar information must be passed to processor *R* for eliminating the (i+7)-th unknown.

The information about this situation is represented for each processor *Q*, *R*, ... separately as follows:

n_rows	= number of equations below the *i*-th with non-zero entries in column *i*. Here n_row = 4.
ibuf	= three items #1 list of row indices of non-zero elements in column *i*. Here the list is i+1, i+2, i+4, i+9 #2 list of column indices of non-zero elements after the first in row *i* associated with processor *Q*. Here the list is i+6, i+8 #3 value of n_rows. Here n_rows = 4
len_ibuf	= length of ibuf array. Here len_ibuf = 7
hdr_proc(j)	= first column (unknown) index in row *i* with a non-zero coefficient whose equation is assigned to processor $j - 1$. It is defined in more detail in Section IV.f below.

Then the content of *ibuf* for the simple example and processor *Q* is

i+1, i+2, i+4, i+9, i+6, i+8, 4

This information is communicated to processor *Q* and then new information is created for processor *R* with len_ibuf = 5:

i+1, i+2, i+4, i+9, 4

and sent to it. This is repeated until all relevant information has been sent to other processors.

IV.f. Auxiliary Information Used in Forming ibuf

This structure contains information about which processors are to receive information by communicating *ibuf*. We have a list of pointers *hdr_proc(j)* for all processors indicating their status with respect to equation i. There are three cases:

Case 1: $hdr_proc = 0$

Processor $j-1$ has no equations whose associated unknowns have non-zero coefficients in the current (i-th) pivot equation. Or processor $j-1$ is processor P ($i = j-1$). In either case no communication is needed.

Case 2: $hdr_proc = -1$

Processor $j-1$ has equations with exactly one associated unknown with non-zero coefficients in the current (i-th) pivot equation. It needs the C-INFO information.

Case 3: $hdr_proc =$ positive integer

Processor $j-1$ has equations with more than one associated unknown with non-zero coefficient in the current (i-th) pivot equation. It needs the C-INFO information and must process several entries in it.

A more complete example is given below.

Processor	0	1	0	1	3	0	1	2	1	2
Column	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9
row i	x	x	0	y	0	0	x	z	x	0

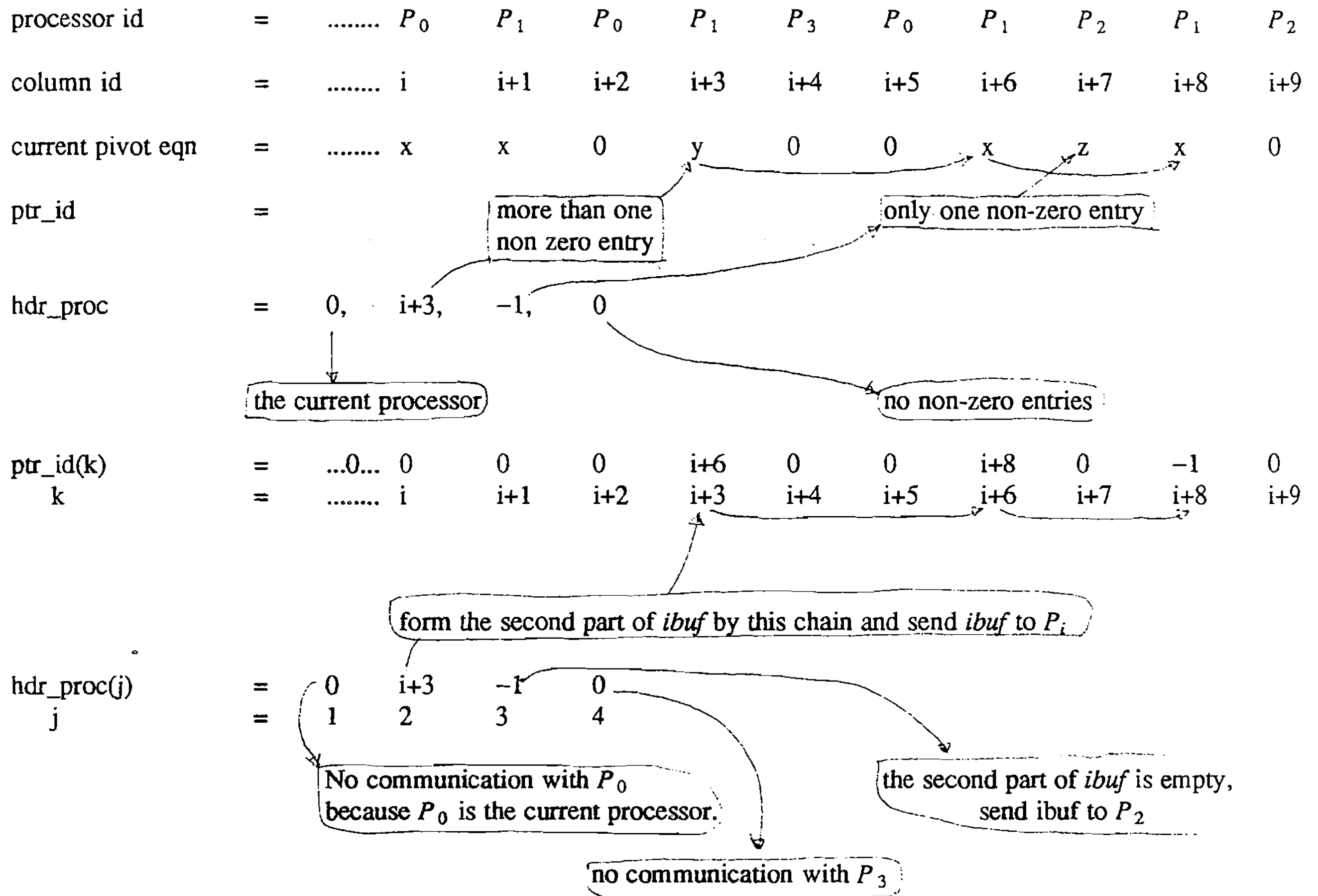
hdr_proc = 0, i+3, −1, 0

Indicated communication status for the 4 processors:

- 0: current processor, no communication
  - 1: has more than one non-zero entry
    Form the second part of *ibuf* with the chain i+3, i+6, i+8, −1 and send *ibuf* to processor 1.
    - 2: has exactly one non-zero entry.
      The second part of *ibuf* is empty, but send *ibuf* to processor 2.
      - 3: has no non-zero entries.
        No communication with processor 3

ptr_id(k)	=	...	0	0	0	0	i+6	0	0	i+8	0	−1	0
k	=	...		i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9

This example is illustrated in a more graphical and detailed form as follows. The algorithm variables are identified explicitly here.



V. STRUCTURE OF THE CODE

The structure of the code PARALLEL SPARSE is described in this section. The hypercube node processor code is decomposed into five levels structurally as shown in Figure 5.1, with each block corresponding to a FORTRAN subroutine or function. Their definitions are described in order as follows.

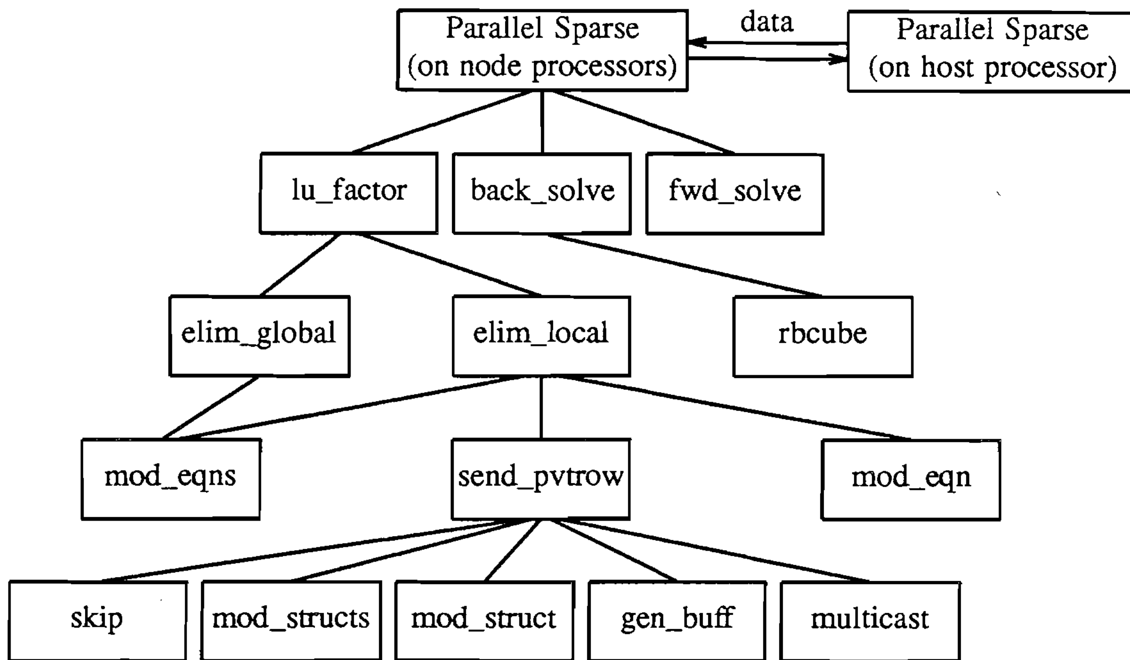


Figure 5.1. The structure of the code PARALLEL SPARSE.

V.a Structure of the Code on the Host Processor

Parallel Sparse. This is the main program which communicates the problem data and the computed results with node processors, and assembles the distributed solution of a sparse linear system of equations.

V.b Structure of the Code on each Node Processor P

Level 1

(1) *Parallel Sparse.* This is the main program which communicates the problem data and the computed results with the host processor, calls subroutines for the LU factorization and for forward and back substitutions to solve a linear system of equations.

Level 2

(2) *lu_factor.* This subroutine carries out Gauss elimination to do LU factorization for the rows in P using the algorithm as described in Section II.

(3) *fwd_solve.* This subroutine performs forward substitution to solve $Ly = b$ for the unknowns in P.

(4) *back_solve.* This subroutine performs backward substitution to solve $Ux = y$ for the unknowns in P.

Level 3

(5) *elim_global*. This subroutine performs the global elimination on a tree node S as described in Section II. Specificly, it checks the dependency of S and its descendents by tracking the array tr in a certain way. For each related descendent T, it further checks each unknown u in T to see if eliminating u has effects on P's remaining rows to be processed. If so, it gets the pivot row of u and performs the modifications.

(6) *elim_local*. This subroutine performs the local elimination on a tree node S as described in Section II. Specificly, it loops over all unknowns u in S. For each u, if necessary, it first participates in communicating the pivot row of u and then performs the modifications on P's rows by the pivot row.

(7) *rbcube*. This subroutine carries out communication (broadcasting) for a message in a hypercube.

Level 4

(8) *mod_eqn* and *mod_eqns*. These two subroutines perform modifications on the first row, in the former, and the remaining ones, in the latter, by a pivot row. The rows to be modified are indicated by a chain consisting of pointers in arrays *hdr_m* and *ptr_m* in M-INFO as described in Section III. The modification task due to a pivot row is here divided into two subroutines because a pipelining technique is also used locally in *elim_local* in order to send the next pivot row out as early as possible.

(9) *send_pvtrow*. This subroutine processes the communication information in C-INFO, generates the pivot row and sends it to appropriate processors according to C-INFO.

Level 5

(10) *mod_struct* and *mod_structs*. These two subroutines update the C-INFO data structures column by column by using the message in array *ibuf* as described in Section IV. The operations are similar to those in *mod_eqn* and *mod_eqns* as above. Again, a pipeline technique is involved here.

(11) *gen_buff*. This subroutine generates the message for array *ibuf* for a processor Q.

(12) *skip*. This subroutine is used in tracking a pointer array *ptr_c* of C-INFO for skipping those entries for which the corresponding C-INFO has been updated.

(13) *multicast*. This subroutine carries out multicasting communication, i.e., sends a message to a set of specified processors (not necessarily to the whole hypercube).

VI. CONCLUSIONS

We present a dynamic data structure to represent general nonsymmetric sparse matrices suitable for Gauss elimination without using a symbolic factorization on a distributed memory machine. Data structures for global computation in a parallel sparse solver based on a block elimination tree are also described. They are used in the module PARALLEL SPARSE as developed for the Parallel Ellpack system. The organization of the code PARALLEL SPARSE is also described. It is efficiently implemented on the NCUBE machine except the multicasting subroutine where we simply use the NCUBE provided primatives *nread* and *nwrite*, which is very inefficient. Our Parallel Ellpack system provides a test bed for the performance evaluation for different software components in solving a PDE problem. PARALLEL SPARSE has been used as a direct *solution* module in the performance evaluation in conjuction with different *assignment* and *indexing* modules. They are reported in [Mu, Rice, 1990a] and [Mu, Rice, 1990b].

REFERENCES

Duff, I.S. (1986), "Parallel Implementation of Multifrontal Schemes", *Parallel Computing*, **3**, pp. 193-204.

George, A., M. Heath, J. Liu, and E. Ng (1988), "Sparse Cholesky factorization on a local-memory multiprocessor", *SIAM Sci. Stat. Comput.*, **9**, pp. 327-340.

George, A., J. Liu, and E. Ng (1987), "Communication reduction in parallel sparse Cholesky factorization on a hypercube", *Hypercube Multiprocessors* (M. Heath, ed.), SIAM Publications, Philadelphia, PA, pp. 576-586.

George, A., J. Liu, and E. Ng, (1988) "A Data Structure for Sparse QR and LU Factors", *SIAM J. Sci. Stat. Comput.*,**9**, pp. 100-121.

George, A., Ng, E (1988), "Parallel Sparse Gaussian Elimination with Partial Pivoting", Technical Report ORNL/TM-10866, Oak Ridge National Laboratory, Oak Ridge, TN.

Houstis, E. and Rice, J.R., (1989), "Parallel Ellpack", *Math. Comp. Simulation*, **31**, (1989), pp. 497-508.

Liu, J. (1988), "Equivalent Sparse Matrix Reordering by Ellimination Tree Rotations", *SIAM J. Sci. Stat. Comput.*, **9**, No. 3, 424-444.

Mu, M. and J.R. Rice (1989a), "LU factorization and elimination for sparse matrices on hypercubes", in *Fourth Conference on Hypercube Concurrent Computers and Applications*, (Monterey, CA, March, 1989), Golden Gate Enterprises, Los Altos, CA, (1990), pp. 681-684.

Mu, M. and J.R. Rice (1990a), "A Grid Based Subtree-Subcube Assignment Strategy for Solving PDEs on Hypercubes", CSD-TR 869, CER-89-12, Computer Science Department, Purdue University, Revised in April, 1990.

Mu, M. and J.R. Rice (1990b), "The structure of Parallel Sparse Matrix Algorithms for Solving Partial Differential Equations on Hypercubes", CSD-TR 976,, CER-90-19, Computer Science Department, Purdue University, May, 1990.