1989

Solving Linear Systems with Sparse Matrices on Hypercubes

Mo Mu

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

Report Number:
89-870

SOLVING LINEAR SYSTEMS WITH SPARSE MATRICES ON HYPERCUBES

Mo Mu
J. R. Rice

CSD-TR-870
February 1989

SOLVING LINEAR SYSTEMS WITH SPARSE MATRICES ON HYPERCUBES

Mo Mu * and John R. Rice**

Computer Science Department
Purdue University
West Lafayette, IN 47907
Technical Report CSD-TR-870
CAPO Report CER-89-11
February 1989

ABSTRACT

We investigate parallel Gauss elimination for sparse matrices, especially those arising from the discretization of PDEs. We propose an approach which combines **minimum degree ordering, nested dissection, domain decomposition** and **multifront** techniques. Neither symbolic factorization nor explicit representation of **elimination trees** are needed. An effective and economic **dynamic data structure** is presented along with a **grid based subtree-subcube** assignment strategy which enhances load balancing, high parallelism and low communication cost. The algorithm is implemented on the NCUBE/7

* Work supported in part by National Science Foundation grant CCR-8619817.
** Work supported in part by the Air Force Office of Scientific Research grants 84-0385, 88-0243.

I. INTRODUCTION

We consider parallel Gauss elimination for sparse matrices with emphasis on those arising from the discretization of partial differential equations (PDEs). This emphasis arises because we need modules in our PELLPACK (parallel ELLPACK) software [Houstis and Rice, 1989]. There have been many good techniques for general sparse sparse matrices (eg., [Duff, Gould, Lescrenier and Reid, 1987], [Geist and Ng, 1988], [George, Heath, Liu and Ng, 1988], [George and Ng, 1988], [Johnson and Dongarra, 1987] and [Rice, 1986]. Most of them are for the Cholesky factorization of symmetric positive definite matrices. Algorithms for general sparse matrices are usually very complicated, [Ng, 1989] surveys the recent achievements in this active field. Matrices from PDE problems usually have special sparse structures even though they may be nonsymmetric. We seek to exploit these structures and still remain able to handle completely general problems. Our approach is to combine well known techniques for sparse solvers, such as **minimum degree ordering, nested dissection, domain decomposition (substructure)** and **multifront**, so as to exploit their advantages and minimize their disadvantages to some degree. We avoid the usual **symbolic factorization** and the explicit representation of the **elimination tree** structure. An effective and economic **dynamic data structure** is used instead. We also suggest a **grid based subtree-subcube** assignment strategy which enhances load balancing, high parallelism and low communication cost on hypercube machines. Some other issues in implementation like pipelining and broadcasting are also considered. The algorithm is implemented on the NCUBE/7.

II. ELIMINATION TREES

Elimination trees have been extensively used in developing parallel sparse solvers. In general, a balanced, short and many branched tree shape leads to good parallelism. The two main factors of elimination trees affecting these characteristics are ordering and lack of symmetry.

In factoring sparse matrices, one wants an ordering which produces low fill-in. Two well known orderings are **minimum degree** and **nested dissection**. The former has the advantage of minimum fill-in, but generates very tall and thin trees, while the latter has moderate fill-in and generates well shaped trees. In addition, nested dissection prefers PDE problems with regular grids and irregular grids could reduce its effectiveness dramatically. There are some efforts on optimizing tree shapes from the minimum degree ordering by reordering, rotating and so on [Lewis, Peyton, Pothen,

1989], [Liu, 1988].

For symmetric positive definite matrices elimination trees are defined by the nonzero structures of their Cholesky factors. If A is a nonsymmetric matrix one usually defines its elimination tree by the structure of the Cholesky factor L_c of A^TA. There is also a modified symbolic factorization which generates a structure for $\overline{L} + \overline{U}$ [George and Ng, 1988]. This structure contains all possible pivotings in actually factoring A into LU with the properties that the structure of L is contained in that of $\overline{U}^T$ and the structure of $\overline{U}$ is contained in that of L_c^T. Then the elimination tree can be defined by $\overline{U}$. It is obvious that this approach could introduce extra, unnecessary fill-in and cause the tree to be unbalanced. Recall in solving PDE problems that lack of symmetry often occurs locally from treating boundary conditions. This local lack of symmetry can affect tree structures globally in the above process.

Of course, the approaches mentioned above are intended for general purposes and effective for general sparse matrices. For the special sparse structures in PDE applications we propose to mix nested dissection with minimum degree using **domain decomposition.** This geometric approach is different from the usual algebraic one in the sense that elimination trees are in condensed form. Each node contains a group of unknowns which are grouped according to the number of processors available. Nested dissection is applied globally and the minimum degree ordering is used locally. Local nonsymmetry does not affect the global tree structures. Also, it is unnecessary to perform symbolic factorization.

For simplicity, we consider solving problems on a rectangular domain Ω. The approach can be easily extended to general domains. Suppose we have $p\ (= 2^{2d})$ processors available. By domain decomposition Ω is divided into p subdomains Ω_{ij}, $i, j = 1, 2, \ldots, p^{1/2}$ as shown in Figure 2.1.

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 2.1. Domain decomposition of a rectangle for $d = 2$.

In principle, one can put a local grid on each subdomain and apply a discretization appropriate for local problem properties. Thus the local solutions U_{ij} only depend on unknowns at grid points of $\partial\Omega_{ij}$. So the standard domain decomposition (or substructure) approach is as follows [Farhat, Wilson, Powell, 1987]. First, all interior unknowns U_{ij} are eliminated locally. This step is obviously totally parallel. Notice that the elimination within each subdomain occurs in a single processor and therefore it is essentially a sequential computation. Any efficient ordering can be used here. So we propose the minimum degree ordering at this step without worrying about its unsatisfactory effect on parallelism. Second, all processors participate in eliminating interface unknowns as in dense solvers. The resulting matrix structure is shown in Figure 2.2.

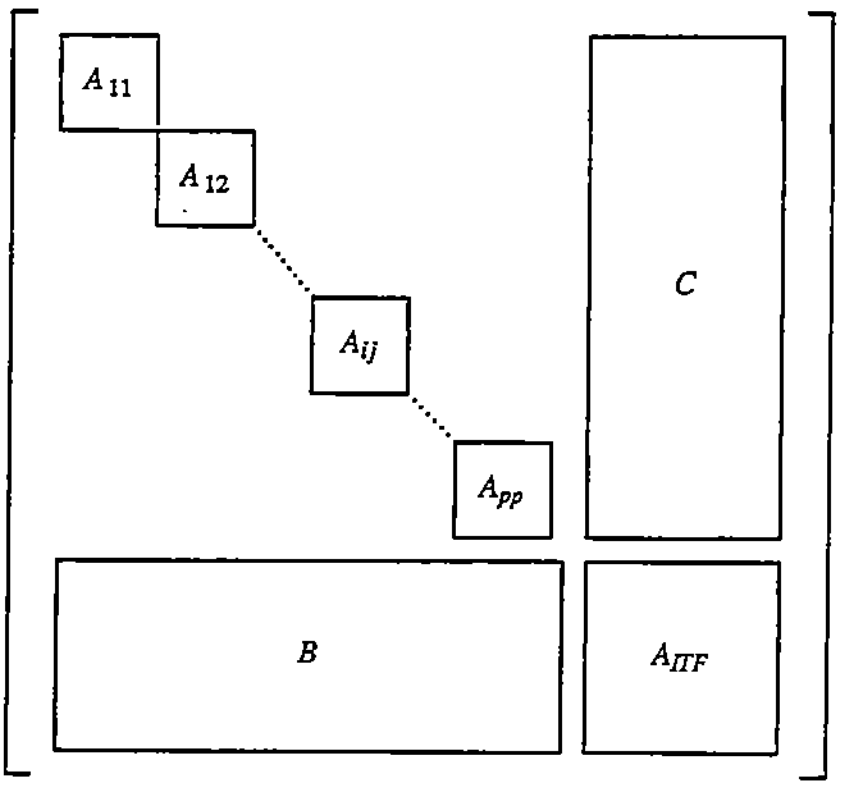

Figure 2.2. General matrix pattern from domain decomposition. The blocks A_{ij} relate the interior unknowns U_{ij} of the problem, B and C link these to the interface unknowns which are related through the matrix A_{ITF}.

We notice that the resulting matrix is block symmetric in some sense due to the domain decomposition technique even though locally symmetry might not occur. However, this

standard domain decomposition does not exploit the further sparse structure for the interface part (B, C, A_{ITF}).

We achieve further efficiency by exploiting the considerable structure within the interface unknowns. The nested dissection approach is applied now since it exploits parallelism well, the number of interface unknowns in the interface set is only of order $O(N^{1/2})$ if the total number of unknowns is N. At this second step parallelism becomes more important than fill-in. We use one way nested dissection to partition the interface set into several levels for a hypercube machine, each level consisting of several groups of nodes in the elimination tree. The partition is illustrated by Figure 2.3 along with the numbering of the elimination tree nodes (groups of unknowns). It is sometimes convenient to treat a group of unknowns in the subdomain interface as a subdomain.

This approach can also be viewed as a variant of incomplete nested dissection. If nested dissection were used for the local ordering within subdomains then this leads to an ordinary nested dissection ordering.

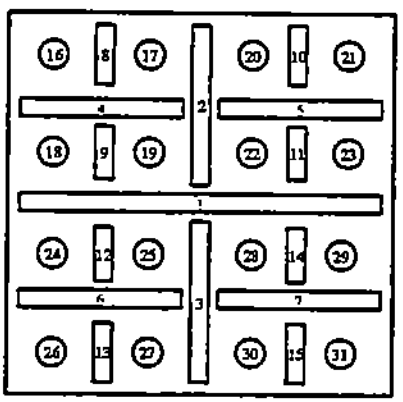

Figure 2.3. Partition of the subdomain interfaces in Figure 2.1 using one way nested dissection. The circles (16–31) represent the 16 groups of interior unknowns and the boxes represent groups of interface unknowns. All boxes of the same size are on the same level of the elimination tree.

The PDE discretization process leads to a local dependence or boundary dependence property for interfaces. For example, if we consider the union of subdomains 16, 17, 8 as a generated subdomain Ω'_8 then the local interior solution set U'_8 is uniquely determined by unknowns at grid points on $\partial\Omega'_8$. This relation holds similarly for groups at higher levels of the elimination tree for the unknowns arising in PDE

applications. Thus we obtain a condensed binary elimination tree as shown in Figure 2.4 which reflects the local dependency.

This elimination tree has the following properties:

(a) Each node corresponds to one subdomain, local ordering and lack of symmetry do not affect the tree structure.

(b) Eliminating a node only has effects on its ascendants.

(c) The eliminations of the descendent nodes of a given node are independent of one another.

Properties (b) and (c) hold for all elimination trees. These properties are used in the parallel *LU* factorization in following sections.

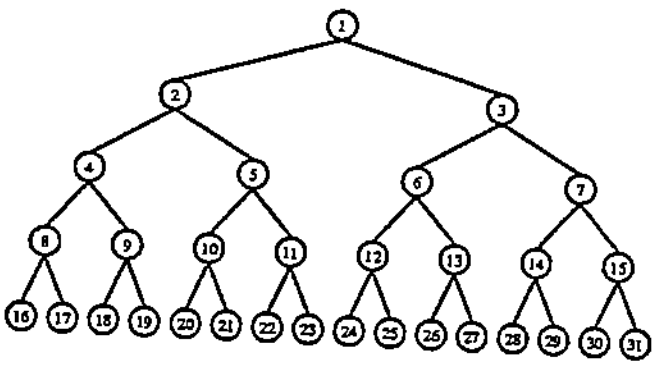

Figure 2.4. Condensed elimination tree produced by domain decomposition and nested dissection. The numbering of nodes corresponds to the groups of unknowns in Figure 2.3.

An advantage to this approach is that the trees generated are so simple that it is not necessary to store their structure. Searching a tree is also very easy using the following relations for node i,

$$\text{left son} = 2i \tag{2.1a}$$

$$\text{right son} = 2i + 1 \tag{2.1b}$$

$$\text{parent} = [i/2] \tag{2.1c}$$

III. PARALLELISM AND ASSIGNMENT STRATEGY

(a) **Potential parallelism.** There are several kinds of potential parallelism here. First, elimination steps in independent nodes can execute simultaneously. By an elimination step we mean those operations for eliminating an unknown, such as local pivoting within the node it resides, generating and broadcasting multipliers, and communicating the fill-in information caused by this elimination step. 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 because each leaf node has only one processor even though it represents a sparse subproblem. For leaf nodes we can choose the most efficient sequential sparse solver based on the **minimum degree ordering.** For the higher level nodes we see that the subproblems become more dense and there are more processors than nodes. So various efficient parallel dense solvers can be applied to exploit the inner parallelism. Third, the modification task in elimination is independent for different equations just as for dense matrices. Finally, fourth, modifications, even on the same equation, due to independent descendent nodes can be performed in arbitrary order.

Local pivoting strategies within nodes can be used without affecting the global parallelism. However, we do not employ pivoting in the following because usually it is not required for systems resulting from the discretization of elliptic problems (it is never necessary for symmetric definite matrices). Our approach could be modified for those occasional problems where local pivoting is required.

(b) **Assignment Strategy.** By assigning a grid point to a processor we mean assigning both the problem data and the factorization subtask associated with this point. That is, the equation at this point is stored in the assigned processor and manipulations in modifying and eliminating the associated unknown are also assigned to the same processor. In order to achieve high parallelism, load balancing and low communication 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 connection. The standard wrapping assignment used for dense matrices is not as effective here even though it achieves load balancing. [George, Liu, Ng, 1987] proposes a **subtree-subcube** assignment which is very satisfactory. To apply their scheme to our condensed trees, we observe that they only consider the global assignment issue. They simply employ **wrap mapping** for assigning a subtree to points within each node. We call this assignment **subtree-subcube with local wrapping.** Recall the following two important facts. First, eliminating an unknown in a node need not affect all of its

ascendant nodes. Second, even when effects occur in some ancestor nodes, they need not affect all equations in them. Geometrically, the effect of elimination spreads in a multifrontal manner. However, one cannot represent these properties completely by elimination trees and yet they may affect the parallel efficiency very much. This observation suggests that grid point assignment should be made in a multifrontal manner. Specifically it is best to have a processor responsible only for those grid points located at the fronts of some nodes to which the processor has been assigned. If several processors (usually a subcube) correspond to the same set of points, then local wrapping can be applied within this set. We call this the **grid based subtree-subcube** assignment which is defined more precisely as follows.

Let us denote the levels in the tree from bottom to top by 0, 1^{st} x, 1^{st} y, 2^{nd} x, 2^{nd}, ..., i^{th} x, i^{th} y and so on. The first step is to map the given hypercube to a two dimensional grid (for domain decomposition) by the well-known *gray code* such that adjacent processors are directly connected. It is natural to assign Ω_{ij} to processor P_{ij}. Next, we subdivide each node on the i^{th} x level and the i^{th} y level into 2^{i-1} and 2^i segments, respectively. This subdivision of segments corresponds to the natural geometric segments in domain decomposition. We take care of the intersection points of adjacent segments in each node by adding an intersection point to its left (top) segment for the $x(y)$ direction. Then we assign each segment on i^{th} $x(y)$ level to the closest 2^i processors in the $x(y)$ direction. The assignment within each segment uses wrapping. This scheme is illustrated in Figure 3.1.

In [George, Liu, Ng, 1987] it is proved that the total amount of communication for **subtree-subcube with local wrapping** is $O(pN)$. This communication order is optimal in the sense of minimizing traffic volume for nested dissection algorithms. This order is also valid for our **grid based subtree-subcube** approach as George's analysis applies for all types of subtree-subcube assignments. If one includes the startup communication cost as well as a cost per item that exist for hypercubes, then our grid based subtree-subcube assignment is much more efficient. We find that the cost for their assignment is $O(p^{3/2}\log_2 p)N^{1/2}s$ while our cost is $O(3/4p\log_2 p)N^{1/2}s$, where s stands for time for startup. So we gain an $O(p^{1/2})$ reduction in startup cost by using the multifrontal idea. This is a substantial gain for current hypercube multiprocessors with costly communication startup. This analysis is given in [Mu, Rice, 1989].

IV. DYNAMIC DATA STRUCTURE

A common approach in recent parallel sparse solvers uses symbolic factorization as preprocessing which is then used for generating elimination trees and allocating static data structures. In our approach the elimination trees are predefined without any symbolic factorization. Also in PDE packages such as PELLPACK, the data structure for *LU* decomposition is used only once in the whole process. Therefore we use a dynamic data structure that mixes the symbolic and numerical computations and which allows tailoring the computation to the problem. This dynamic data structure can be arranged to have minimal cost for selected problems and in addition, it has very economical storage requirements.

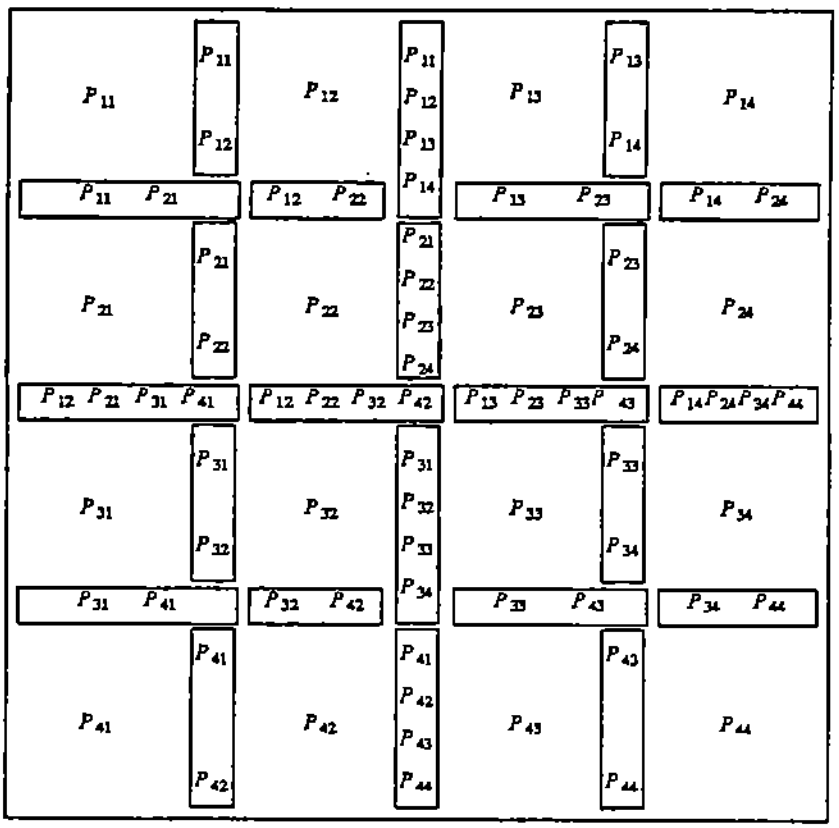

Figure 3.1. Grid based subtree-subcube assignment for 16 processors. Within the subdomain interfaces we show how the processors are assigned to parts of the groups of grid points.

The data structure has three main parts.

(a) **Part 1: Sparse matrix information.** This part represents the matrix $A = (A_{ij})$ and its factorization $A = LU$. Part 1 consists of five vectors in a particular processor P:

hdr_rowu:

dimension	=	number of equations assigned to P
hdr_rowu(i)	=	pointer header for P's ith row in U

hdr_rowl:

dimension	=	number of equations assigned to P
hdr_rowl(i)	=	pointer header for P's ith row in L

a:

dimension	=	number of nonzeros of the factors of A stored in P
a	=	list used for storing nonzero entries of matrix A

id_col:

dimension	=	dimension of a
id_col(i)	=	the column index of the entry in $a(i)$

ptr_a:

dimension	=	dimension of a
ptr_a(i)	=	the location of the next nonzero entry in the same row, i.e., if $a(i) = A_{kl}$ and the next nonzero in row k is $A_{kl'}$, then $a(ptr\_a(i)) = A_{kl'}$. We have $ptr\_a(i) = 0$ iff A_{kl} is the last nonzero entry in row k.

(b) **Part 2: Modification Information.** This part provides the structure describing the modification subtask for an individual processor. It is dynamically updated due to new fill-in during the elimination process and consists of four vectors.

hdr_m:

dimension	=	number of equations of the system
hdr_m(i)	=	pointer to header for P's equations depending on i^{th} unknown

ptr_m:

dimension	=	number of nonzeros of L stored in P
ptr_m(j)	=	pointer to the next equation depending on the same unknown. $pr\_m(j) = 0$ iff the end of the chain of equations depending on the unknown is reached.

idg_m:

dimension	=	dimension of ptr_m
idg_m	=	list of global indices of equations to be modified

hdrad_m:

dimension	=	dimension of ptr_m
hdrad_m	=	list of pointers to beginning nonzero entries to be updated in equations to be modified

(c) Part 3: Communication Information. This part defines the communication requirements for individual processors. It is also dynamically updated due to fill-in during the elimination process. Processor P has three vectors:

hdr_c:

dimension	=	number of equations assigned to P
hdr_c(i)	=	pointer to header for nonzeros in column I, if P's ith equation has global index I

ptr_c:

dimension	=	number of nonzeros in those columns of the factored matrix whose unknowns are assigned to P.
ptr_c(i)	=	pointer to the next nonzero in the column, $ptr\_c(i) = 0$ iff this is the last nonzero in this column

idg_c:

dimension	=	dimension of ptr_c
idg_c	=	list of row indices of nonzeros pointed by the corresponding pointers in ptr_c.

The order of the total storage requirement in this data structure is $6.5N \log_2 N/p + 6p$.

To show the dynamic data structure operation consider

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} & 0 & a_{17} \\ a_{21} & a_{22} & 0 & a_{24} & 0 & 0 & a_{27} \\ 0 & a_{32} & a_{33} & 0 & a_{35} & a_{36} & 0 \\ a_{41} & 0 & a_{43} & a_{44} & 0 & 0 & a_{47} \\ 0 & 0 & 0 & a_{54} & a_{55} & 0 & a_{57} \\ a_{61} & a_{62} & 0 & a_{64} & 0 & a_{66} & 0 \\ 0 & 0 & a_{73} & 0 & 0 & a_{76} & a_{77} \end{bmatrix}$$

Suppose rows 1 and 6 are assigned to processor P, then the original matrix information in P is as follows.

vector: $a = a_{11}, a_{12}, a_{15}, a_{17}, a_{61}, a_{62}, a_{64}, a_{66}$

vector: id_col = 1, 2, 5, 7, 1, 2, 4, 6

vector: ptr_a = 2, 3, 4, 0, 6, 7, 8, 0

vectors: hdr_rowu = 1, 8, hdr_rowl = 1, 5

Fill-in during elimination just requires extending a and updating id_col and ptr_a. The information for modification and communication reads:

vector: idg_m = 6, 6, 6, ···

vector: ptr_m = 0, 0, 0, ···

vector: hdrad_m = 5, 6, 7, ···

vector: hdr_m = 1, 2, 0, 3, 0, 0, 0

vector: idg_c = 1, 2, 4, 6, 3, 6, 7

vector: ptr_c = 2, 3, 4, 0, 6, 7, 0

vector: hdr_c = 1, 5

This information is also dynamically updated.

V. ALGORITHM DESCRIPTION

We now give an informal description of our algorithm.

Algorithm

* Algorithm for processor P for the LU factorization using mixed minimum

* degree, nested dissection by domain decomposition, and grid based

* subtree-subcube assignment of processors

1. Local elimination in the leaf node assigned to P by minimum degree based sparse solver

2. For $\ell = 1$ to $2d$

    2.1. Let *nodel* = the node on ℓ-th level which is assigned to the subtree which contains processor P.

    2.2. Complete processor P modification subtasks in *nodel* due to eliminating all of descendant nodes of *nodel*.

    2.3. Perform local elimination in *nodel* by any parallel dense solver.

end ℓ loop

Here levels in the tree are numbered from bottom to top by 0, 1, 2, ..., $2d$ with level 0 corresponding to subdomains at the leaves. By local elimination in steps 1 and 2.3 we mean performing computations to eliminate unknowns local to the current node and modifications of the associated local equations. Only those equations in processor P's ancestor nodes are modified where associated multipliers are available due to manipulating local equations. The philosophy here is, on one hand, the **pipelining** idea because it tries to start elimination for each step as soon as possible in order to reduce the waiting time in other processors for this message. On the other hand, it is the **multifrontal** idea because, for those equations where the elimination front has not arrived, there is no necessity to immediately process the effects on them due to elimination steps of its descendant nodes. This process causes communication in higher levels of the subcube which are unnecessary for the current step. These communication and modification tasks are delayed to step 2.2. This approach restricts communication, at each step, to as small a subcube as possible.

There are similar **pipelining** tricks used for the elimination steps and manipulating communication information in the local elimination phases. In addition, the *broadcast* issue for communication in sparse solvers is different from that in dense solvers. In the dense case, a very effective **bcube** algorithm can be used. For the sparse case, however, broadcasting usually is not need to the whole subcube. Several of the general purpose **multicasting** algorithms proposed for hypercubes can be applied to sparse solvers e.g., [Ho, Johnsson, 1986]. We can hope to develop even more effective broadcasting algorithms for these applications since broadcasting is done in special ways even though it is not in a complete subcube broadcasting.

This algorithm has been implemented in our NCUBE/7 and runs successfully. We do not present experimental data here because we have not collected enough data to show its performance characteristics and to compare it with other approaches. These data will be reported later.

References

Duff, I., N. Gould, M. Lescrenier, J. Reid (1987), "The multifrontal method in a parallel environment", Computer Science and Systems Division, Harwell Laboratory, Oxfordshire, England.

Farhat, C., E. Wilson, and G. Powell (1987), "Solution of finite element systems on concurrent processing computers", *Engr. Comput.*, 2, 157-165.

Geist, G. and E. Ng (1988), "A partitioning strategy for parallel sparse Cholesky factorization", Tech. Rept., ORNL/TM-10937, Oak Ridge National Laboratory, Oak Ridge, TN.

George, A., M. Heath, J. Liu, and E. Ng (1988), "Sparse Cholesky factorization on a local-memory multiprocessor", *SIAM Sci. Stat. Comput.*, vol. 9, no. 2, 327-340.

George, A., M. Heath, J. Liu, and E. Ng (1988), "Solution of sparse positive definite systems on a hypercube", Tech. Rept., ORNL/TM-10865, Oak Ridge National Laboratory, Oak Ridge, TN.

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, 576-586.

George, A. and E. Ng (1988), "Parallel sparse Gaussian elimination with partial pivoting", Tech. Rept., ORNL/TM-10866, Oak Ridge National Laboratory, Oak Ridge, TN.

Ho, C., and L. Johnsson (1986), "Distributed routing algorithms for broadcasting and personalized communication in hypercubes", *Int. Conf. Parallel Proc. IEEE*, 640-648.

Houstis, E. and J. Rice (1989), "Parallel ELLPACK", *Math. Comp. Simul.*, 31.

Johnson, S. and J. Dongarra (1987), "Solving banded systems on a parallel processor", DCS/TR-519, Department of Computer Science, Yale University.

Lewis, J., B. Peyton, and A. Pothen (1989), "A fast algorithm for reordering sparse matrices for parallel factorization", Tech. Rept., ORNL/TM-110400, Oak Ridge National Laboratory, Oak Ridge, TN.

Liu, J. (1988), "Equivalent sparse matrix reordering by elimination tree rotations",

SIAM J. Sci. Stat. Comput., vol 9, no. 3, 424-444.

Mu, M. and J. Rice (1989), "A grid based subtree-subcube assignment strategy for solving PDEs on hypercubes", CSD-TR-869, Computer Science Department, Purdue University.

Ng, E. (1989), "Parallel direct solution of sparse linear systems", Tech. Rept., ORNL/TM-11045, Oak Ridge National Laboratory, Oak Ridge, TN.

Rice, J. (1986), "Parallel methods for PDEs", in *The Characteristics of Parallel Algorithms* (Jamieson, Gannon, Douglass, eds.) Chapter 8, MIT Press, 209-231.