Parallel Computing 5 (1987) 219–246
North-Holland

Solving banded systems on a parallel processor *

Jack J. DONGARRA and Lennart JOHNSSON

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844, and Computer Science Department, Yale University, New Haven, CT 06520, U.S.A.

Abstract. In this paper we examine ways of solving dense, banded systems on different parallel processors. We start with some considerations for processors with vector instructions, then discuss various algorithms for the solution of large, dense, banded systems on a parallel processor. We analyze the behavior of the parallel algorithms on distributed-storage architectures configured as rings, two-dimensional meshes with end-around connections (tori), boolean n-cube configured architectures, and bus-based and switch-based machines with shared storage. We also present measurements for two bus-based architectures with shared storage, namely, the Alliant FX/8 and the Sequent Balance 21000.

Keywords. Solving large, dense banded systems, parallel processors, distributed-storage architectures, performance measurements, Alliant FX/8, Sequent Balance 21000.

1. Introduction

The solution of banded systems of equations is important in many areas of scientific computing. We restrict our attention here to matrices that are symmetric positive definite or diagonally dominant. With the coming of parallel processors, algorithms for the solution of these problems have been the subject of many studies [1,8,25,28,32]. In this paper we look at an implementation that uses a partitioning scheme essentially equivalent to incomplete nested dissection. This approach divides the work into sections, reduces the sections independently in parallel, performs nearest neighbor communication among processors to compute part of the solution, and carries out independent operations to determine the complete solution to the original problem.

2. Sequential case

LINPACK [5] includes routines to solve banded systems of linear equations. The matrices can be of a general nature or symmetric positive definite. The algorithms for the banded problem in LINPACK are based on the vector operation from the BLAS [29] called a SAXPY, $(y \leftarrow y + \alpha x)$. This operation forms the computational kernel in the form of a rank-1 update of a submatrix of size m^2 for a banded matrix A of order N and half bandwidth m during the kth step of the reduction.

* Work supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and in part by the Office of Naval Research under Contract N00014-84-K-0043.

Table 1
Performance for the CRAY X-MP-1 (order 5000, symmetric positive definite matrix)

Half bandwidth	LINPACK CAL BLAS	Mflops improved FORTRAN MV	Improved CAL MV
8	4	5	11
16	6	12	27
22	7	17	41
44	12	40	84
62	15	54	108
128	26	86	141
200	38	103	149

Since much data are updated at each step, this algorithm suffers in terms of performance on most sequential computers. This effect of poor performance is most pronounced on vector architectures that have vector registers. On such machines the largest bottleneck is in memory traffic.

To substantially reduce memory accesses, the algorithm can be reorganized using matrix-vector multiplication as the computational kernel, instead of rank-1 updates [6,7]. This technique reduces the number of storage accesses from $O(Nm^2)$ to $O(Nm)$ through vectorization.

Table 1 shows the performance differences from this approach.

We can exploit a modest amount of parallelism by performing, say, the matrix-vector operations in terms of independent inner products. An additional amount of parallelism can be exploited by factorizing the matrix from both the top and the bottom simultaneously. When the factorization reaches the middle, a small block matrix must be factored. The backsubstitution then proceeds from the middle and works toward the top and bottom in parallel. It is important to note that the number of floating-point operations is exactly the same in this approach as in the conventional factorization. (This two-way factorization resurfaces from time to time. Jim Wilkinson describes this as an approach used in the early days of computing to reduce overhead associated with the looping [4]. LINPACK uses this algorithm to solve symmetric positive definite tridiagonal systems of equations. Evans and Hatzopolus [10] describe the algorithm as a folding technique.)

We consider this algorithm as the best we can do on a sequential computer. The amount of work, in terms of operations performed, is $m(m+1)(N-(2m+1)/3)$ additions and multiplications for the factorization of symmetric positive definite matrices and $m^2(2N-1)-(4m^2-1)m/3$ for an arbitrary banded matrix of size N and half bandwidth m. The number of additions/subtractions and multiplications for each solve is $2m(2N-m-1)$.

3. Parallel algorithms

For computer architectures that provide a large number of processors, algorithms can be employed that exploit the independence of operations in eliminating single variables and the independence of operations in eliminating different variables [12,14,19,21,22,30,42,43]. For a narrow banded matrix the potential concurrency is largely due to the independence of operations in eliminating different variables, whereas for a matrix with a half bandwidth equal to $\sqrt{N}$ or larger the concurrency due to the independence of operations in the elimination of a single variable dominates.

If only the complexity of the parallel arithmetic is considered, as would be reasonable for a shared-memory machine with very high communication bandwidth which allow conflict-free access to data, then the speedup from concurrent elimination of single variables is linear. The

speedup is measured as the time for sequential elimination divided by the time for concurrent elimination. On the other hand, the speedup from exploiting the independence of operations in eliminating different variables always is sublinear, since more operations are required to find the solution. The maximum speedup in the latter case is in the range $1/2(N/m)/\log(N/m)$ to $(N/m)/\log(N/m)$ for N/m processors, with a minimum efficiency of $O(1/\log(N/m))$. In the former case the maximum speedup is m^2, with an efficiency equal to 1 [23].

The graph representation of a banded matrix constitutes a *perfect elimination graph* [34], and Gaussian elimination without pivoting constitutes a perfect elimination order. Hence, there is no *fill-in*. Concurrent elimination of different variables, on the other hand, generally does not constitute a perfect elimination order; the total number of arithmetic operations performed and the storage required by the concurrent algorithm are higher than for the sequential algorithm. However, the time to solve the banded system should be greatly reduced for sufficiently many processors. In the case of a tridiagonal system, odd-even cyclic reduction requires approximately $17N$ operations compared to $8N$ for Gaussian elimination, but the parallel arithmetic complexity for cyclic reduction is $11 \log N$ in $2 \log N$ steps (or $12 \log N$ in $\log N$ steps). Classical Gaussian elimination is a sequential method and required $8N$ operations in $2N$ steps. The two-way Gaussian elimination is also a perfect elimination order. The partial arithmetic complexity is $4N$ during N steps.

Unfortunately, for many architectures, ignoring the cost of communication and the effects of vector features and storage hierarchy is not a good approximation of reality. Communication bandwidth, overhead in communication, routing conflicts, and bank conflicts, as well as overhead for vector instructions (should such be available), are important factors in choosing an optimum algorithm. We will focus on the communication issue. We carry out a simple analysis for two algorithms exploiting the independence of operations in eliminating a single variable, and one algorithm exploiting the independence of operations in eliminating different variables. We discuss ring, mesh, hypercube and shared-memory architectures.

4. Concurrent elimination of a single variable

4.1. One-dimensional partitioning

A one-dimensional partitioning of the matrix in the column or row direction is feasible in the case of a ring of processors, as shown in Fig. 1.

The partitioning can be done either *cyclically* or *consecutively* [27]. In cyclic partitioning by columns, column j is allocated to processor $j \bmod P$, where P is the number of processors labeled from 0 to $P-1$. In consecutive partitioning, column j is assigned to processor $(jP/(m+1)) \bmod P$ (assuming for simplicity that $m+1$ is a multiple of P). Cyclic and consecutive partitionings are illustrated in Fig. 2.

Consecutive partitioning is akin to a block-oriented algorithm. It leads to a lower utilization of the array, but requires fewer communications in a packet-oriented mode of operation where a block corresponds to a packet.

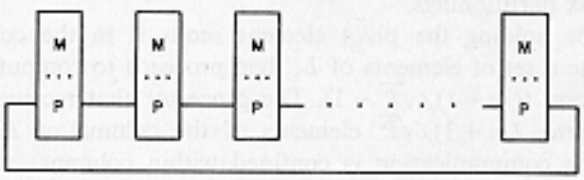

Fig. 1. A ring of processors with distributed storage.

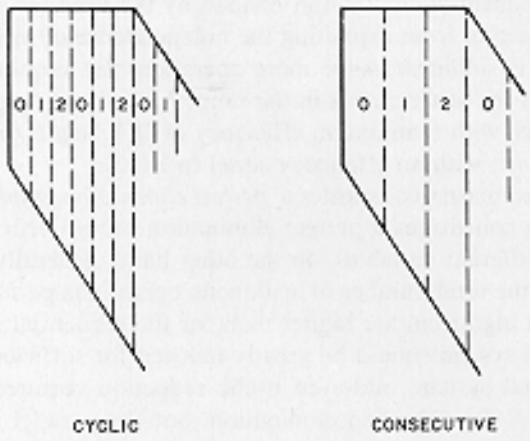

Fig. 2. Cyclic and consecutive one-dimensional partitioning.

In cyclic partitioning a processor first computes the pivot column of the left factor L, then performs its part of the rank-1 update of the m by m submatrix; in other words, it updates $(m+1)/P-1$ columns of length m. After the processor storing the pivot column has computed a new column of L, it transmits the column to the processor with the next higher index modulo P (to the right). The receiving processor forwards the column to the next processor, and performs a rank-1 update on $(m+1)/P$ columns of length m. Assuming a time t_a for addition/subtraction and multiplication and a time t_d for division, and no vector features in the processors, the arithmetic complexity is approximately $(t_d+2(m+1)/P+1)mt_a)N$ for a nonsymmetric matrix. The communication complexity is approximately $(mt_c+\tau)N$, τ is the communications overhead (startup) and t_c is the transmission time for a floating-point number. We assume that the architecture is of the MIMD type [11] and that communication and arithmetic operations are pipelined. The speedup is approximately P if only the arithmetic complexity is considered, but may be poor if the communication complexity dominates. That is, if $(mt_c+\tau)N \gg (t_d+(2(m+1)/P+1)mt_a)N$, then $P \gg 2m^2t_a/(mt_c+\tau)$.

In consecutive partitioning the processor holding the pivot block, factors it, and transmits the factor to the 'next' processor in a way analogous to the cyclic partitioning. The arithmetic complexity is approximately $(t_d+(3(m+1)/P-P)(2m+1)/2Pt_a)N$ for an unsymmetric matrix and the communication complexity $(mt_c+(P/m)\tau)N$. Pipelining of arithmetic and communication operations is assumed for these complexity estimates. The arithmetic complexity is approximately 1.5 times that of cyclic partitioning for $P \ll m$, but the number of startups in communication is reduced by a factor of P/m.

4.2. Two-dimensional partitioning

In the case of a two-dimensional mesh-configured set of processors, a two-dimensional partitioning of the matrix can be beneficial [25]. This partitioning can also be performed cyclically or consecutively. We assume for simplicity that the matrix has the same number of super- and subdiagonals, and that the processor mesh has $\sqrt{P}$ nodes in each coordinate direction. Figure 3 illustrates the matrix partitionings.

In cyclic partitioning the processor holding the pivot element sends it in the column direction (or row direction) to compute a set of elements of L, then proceeds to compute the elements of a column of L that it stores $((m+1)/\sqrt{P}-1)$. The processor that receives the pivot element forwards it and computes $(m+1)/\sqrt{P}$ elements of the column of L that corresponds to the pivot element. The communication is confined within columns. As the

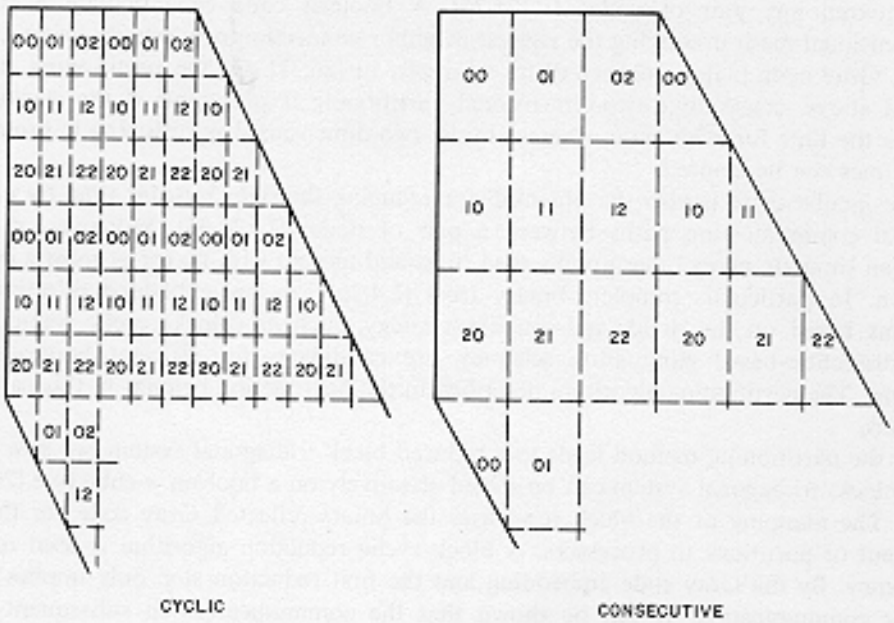

Fig. 3. Cyclic and consecutive two-dimensional partitioning.

elements of L are computed, they are sent to the 'next' processor within the row. The mesh of processors performs a rank-1 update; each processor updates $(m+1)^2/P$ elements. The arithmetic complexity for the cyclic partitioning is approximately $(t_a + (2(m+1)/\sqrt{P}+1)(m+1)/\sqrt{P}\,t_a)N$ for an unsymmetric matrix, and the communication complexity is $((m+1)/\sqrt{P}\,t_c + 2\tau)N$ assuming concurrent communication in the row and column directions. If communication is restricted to one direction at a time, then the data transfer time is doubled and there is a total of N additional startups. The arithmetic complexity is approximately the same as in the one-dimensional cyclic partitioning, but the communication complexity is different. The number of elements transmitted over a channel between a pair of processors is reduced by a factor of $\sqrt{P}$, but the number of communication startups is doubled.

In the case of two-dimensional consecutive partitioning the processor holding the pivot block factors it and passes the left factor to the next processor (right) in the same row, and the right factor to the next processor (down) in the same column. The receiving processor forwards the factors and performs a forward solve. Then a block rank-1 update, a rank $(m+1)/\sqrt{P}$ update, is performed by all processors. The arithmetic complexity of this block algorithm is approximately $(t_a + (11(m+1)^2/P - 9(m+1)/\sqrt{P}+1)3t_a)N$ and the communication complexity $((3(m+1)/\sqrt{P}-1)/2t_c + 2\tau\sqrt{P}/(m+1))N$. The arithmetic complexity is approximately twice (11/6) that of cyclic partitioning. The number of startups is reduced by a factor of $(m+1)/\sqrt{P}$ compared to the two-dimensional cyclic partitioning, and by a factor of $\sqrt{P}/2$ compared to the one-dimensional consecutive partitioning. The time for transfer of matrix elements is approximately 1.5 times that of the two-dimensional cyclic partitioning.

4.3. Boolean n-cube

In a boolean n-cube multiprocessor [38] the processing nodes form the corners of a n-dimensional boolean cube. Each node has a fanout of n; the diameter is n, the average distance between nodes $n/2$, the total number of edges $n2^{(n-1)}$, and the number of disjoint

paths between any pair of nodes n [25,36]. A boolean cube can simulate a ring and two-dimensional mesh preserving the nearest-neighbor connection property by using a binary-reflected Gray code [35], as observed, for example, in [26,31]. Of the partitioning strategies discussed above, consecutive two-dimensional partitioning is preferable if the startup times dominate the time for arithmetic, whereas cyclic two-dimensional partitioning is preferable if startup times can be ignored.

In the n-cube there is also the potential for reducing the data transfer time by using the additional communication paths between a pair of nodes [17,24,36]. Moreover, a boolean n-cube can simulate many other graphs than rings and meshes with no (or at most a constant) slowdown. In particular, complete binary trees [2,4,26] can be embedded effectively, and algorithms based on the divide-and-conquer strategy, such as (block) cyclic reduction and nested dissection-based elimination schemes, are candidates for efficient boolean n-cube algorithms. The partitioning algorithm described in the next section belongs in this category of algorithms.

Since the partitioning method leads to a reduced block tridiagonal system, we now indicate how a (block) tridiagonal system can be solved effectively on a boolean n-cube (see [20,26] for details). The mapping of the block rows uses the binary-reflected Gray code for the initial assignment of partitions to processors. A block cyclic reduction algorithm is used to exploit concurrency. By the Gray code embedding and the first reduction step only implies nearest-neighbor communication. It can be shown that the communication in subsequent steps is always between processors at distance 2 from each other. This property can be used for an *in-place* algorithm. It is also possible to divide the distance 2 communication into two nearest-neighbor communications such that after an exchange step the equations participating in the next reduction step are in a subcube of half size. Hence, during the reduction process, subsets of equations are recursively assigned to subcubes such that any subset is mapped to a subcube in a binary-reflected Gray code order. The recursive assignment requires a simple exchange operation between certain adjacent processors. Each processor can determine whether it will perform an exchange operation and with what processor from its address and also the reduction (backsubstitution) step to be performed. The same local information suffices to determine the communication for the reduction (backsubstitution) operations themselves. The exchange algorithm moves even equations to even processors (and odd equations to odd processors). The considered bit-field is reduced by one for each reduction step. One step of the exchange algorithm is illustrated for a 3-cube in Fig. 4.

In the n-cube algorithms based on binary-reflected Gray codes, full advantage can be taken of truncated cyclic reduction. Each reduction step is carried out on all relevant equations during the same time step. Hence, truncating the reduction after $r < n$ steps reduces the total time proportionally.

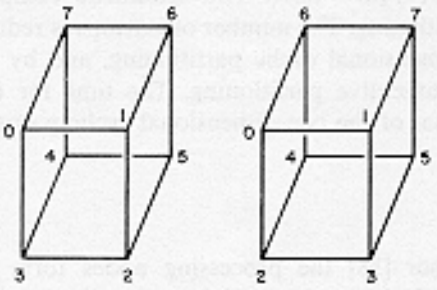

Fig. 4. Recursively assigning equations to nodes through exchange operations preserving adjacency.

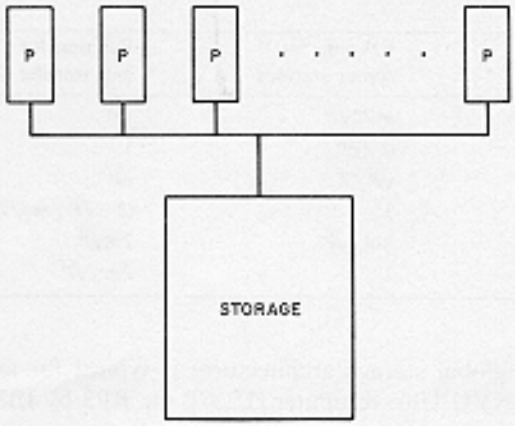

Fig. 5. Global-storage architecture with a bus.

The block cyclic reduction algorithm as outlined above exploits the independence of operations in the elimination of variables corresponding to different blocks. It is also possible to exploit the independence of operations for the elimination of a single variable if additional processors are available. A two-dimensional mesh is suitable for the latter form of concurrency. By assigning $\log 3m^2$ dimensions of the boolean cube for the embedding of $3m \times m$ meshes and $\log P$ dimensions for the partitions, both forms of concurrency can be maximally exploited.

We will analyze the complexity of the partitioning method in some detail in Section 5. For further details see [23,26].

4.4. Global storage architectures

For global storage architectures there are two basic approaches: the use of a common bus for processor-to-storage communication (Fig. 5), and switch-based architectures (Fig. 6).

The bus type of global-storage architecture is typical when the number of processing elements in few, as in the CRAY's, Sequent Balance 21 000, Encore Multimax, and Alliant

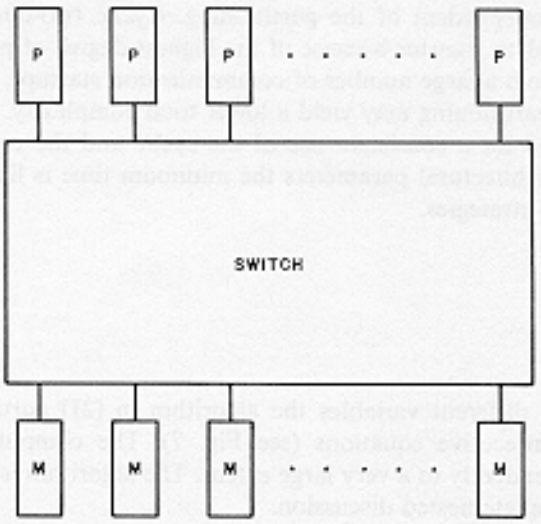

Fig. 6. Global-storage architecture with a switch.

Table 2

Partitioning	Rel. time for arithmetic	Rel. time for comm. startups	Rel. time for data transfer
Cyclic 1-d	1.0	$m/2\sqrt{P}$	$\sqrt{P}$
Cyclic 2-d	1.0	$m/\sqrt{P}$	1
Consecutive 1-d	1.5	$\sqrt{P}/2$	$\sqrt{P}$
Consecutive 2-d	2	1	$(3-\sqrt{P}/m)/2$
Global storage, bus	1.0	$2m/\sqrt{P}$	$2m\sqrt{P}$
Global storage, switch	1.0	2	$2m/\sqrt{P}$

FX/8 computers. The switch type of global storage architectures is typical for architectures conceived as highly parallel, as in the NYU Ultracomputer [15,37], the RP3 by IBM [33], and the BBN Butterfly [3].

In an architecture with global storage and limited local storage the computations for exploting the independence of operations in the elimination of a single variable can be organized such that the processors first share in the computation of a new column of L, then share the computation for the rank-1 update. The arithmetic complexity is $(t_d + (2m + 1)m/Pt_a)N$. The communication complexity for the bus architecture is $(\max((m+1)^2/Pt_{cp}(m+1)^2 t_{cs}) + t_{cp} + (t_{cs}/t_{cp}P + 1)\tau)2N$, where t_{cp} is the time to transfer one floating-point number to or from a processor and t_{cs} the same time for the storage. For the switch-based architecture the communication time is $((m+1)^2/Pt_c 2\tau)2N$. More accurate models of the Alliant FX series are presented by Jalby and Meier [18], who also describe efficient dense matrix routines for that architecture. Sorensen [40] has devised effective matrix routines for the HEP [9], which is a switch-based architecture with local storage.

4.5. Complexity analysis for the concurrent elimination of a single variable

The complexity estimates for the concurrent elimination of a single variable on different architectures can be summarized as in Table 2.

The conclusion from the analysis so far is that for the distributed storage architectures the cyclic partitioning yields the lowest arithmetic complexity because of less idle time. The total number of arithmetic operations is independent of the partitioning. Cyclic two-dimensional partitioning has the smallest time for data transfer because of the highest degree of pipelining. However, cyclic partitioning suffers from a large number of communication startups, and with significant startup times consecutive partitioning may yield a lower total complexity.

The algorithms have been described for a consistent use of the cyclic and the consecutive storage schemes. For a given set of architectural parameters the minimum time is likely to be achieved by a combination of the two strategies.

5. Concurrent elimination of variables

For the concurrent elimination of different variables the algorithm in [21] partitions the system of equations into sets of consecutive equations (see Fig. 7). The computations in different sets can be performed independently to a very large extent. The algorithm is a variant of substructured elimination or incomplete nested discussion.

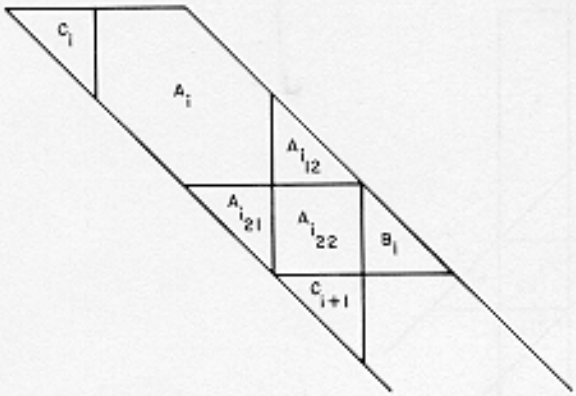

Fig. 7. Partitioning of the matrix.

The algorithm proceeds in four phases:

Phase 1: Factor each partitioned matrix.

Phase 2: Apply factors to pieces of the matrix to decouple the solution.

Phase 3: Form a reduced matrix, and solve this matrix problem forming part of the solution.

Phase 4: Backsubstitute to determine the remaining parts of the solution.

Each of these phases is discussed in detail in the following subsections.

5.1. Phase 1

In Phase 1, each section of the matrix (block $A_{i_{11}}$) is decomposed. This phase can be carried out on each section totally independently of the other sections; there is no communication between sections. At the end of this phase each section has the LU decomposition as part of the matrix. The amount of work for each section is $m(m+1)(k-(2m+1)/3)$ addition and multiplications for symmetric positive definite matrices and $m^2(2k-1)-(4m^2-1)m/3$ for an arbitrary banded matrix of size k and half bandwidth m, where k is the order of the matrix $A_{i_{11}}$.

5.2. Phase 2

In Phase 2, the factors generated in Phase 1 are applied to the matrix. This can be viewed as premultiplying each section of the matrix in Fig. 7 by $\mathrm{diag}(A_{i_{11}}^{-1}, I_{i_{22}})$. The resulting matrix has fill-in from $G_i \leftarrow A_{i_{11}}^{-1}C_i$ and $F_i \leftarrow A_{i_{11}}^{-1}A_{i_{12}}$. The fill-in is diagrammed in Fig. 8. Again in this stage there is no communication between sections. The operation count for this phase is $2m(2k-m-1)(2m+r)$ for r right-hand sides.

5.3. Phase 3

Phase 3 has two parts. First, the system is decoupled and a reduced system is solved, forming part of the original solution. This part of Phase 3 involves zeroing submatrix $A_{i_{21}}$. Zeroing can be accomplished by simple block elimination with the block above $A_{i_{21}}$. This results in some fill-in below G_i of the form $G_i' \leftarrow G_i' - A_{i_{22}}G_i$ and a modification of $A_{i_{22}}$ such that $A_{i_{22}} \leftarrow A_{i_{22}} - A_{i_{21}}F_i$. These operations are independent for each i and can proceed in parallel for all sections. No communication is required.

The second part of Phase 3 involves zeroing B_i using the block from below. This results in a full-in above F_{i+1} of the form $F_{i+1}' \leftarrow F_{i+1}' - B_iF_{i+1}$ and modification of $A_{i_{22}}$ such that

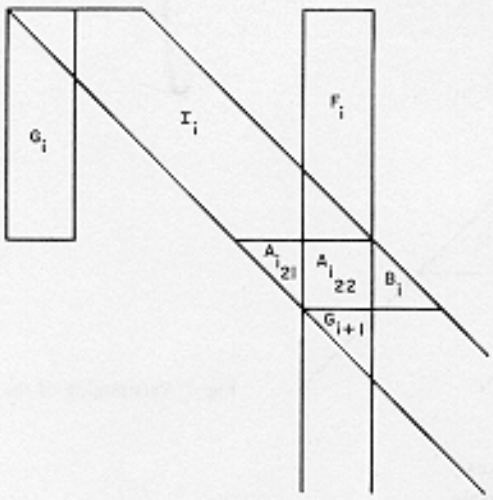

Fig. 8. Matrix after Phase 2.

$A_{i_{22}} \leftarrow A_{i_{22}} - B_i G_{i+1}$. Here interpartition communication must take place to perform the update of block $A_{i_{22}}$ and forming of blocks F_{i+1}.

At this point the matrix has the form shown in Fig. 9.

The last set of m rows of each partition together form a system of mP equations in the last m variables of each partition. The matrix has been decoupled, and a reduced block tridiagonal matrix can be formed. When this block tridiagonal system is solved, the partial solution to the original matrix problem at these positions are formed.

The amount of work for this portion of the algorithm depends on the size of the block tridiagonal system and the method used to solve the system. The blocks are of size $m \times m$, and there will be P block rows, where P is the number of partitions made in the original matrix. If we fix the order of the original matrix and look at what happens as we increase the number of partitions, we see that more effort will go into solving the reduced system.

The communication complexity and the parallel arithmetic complexity for the solution of the block tridiagonal system depend on the architecture [26]. The number of block row communications in sequence depends on the method chosen and the architecture. It varies from $\alpha \log P$,

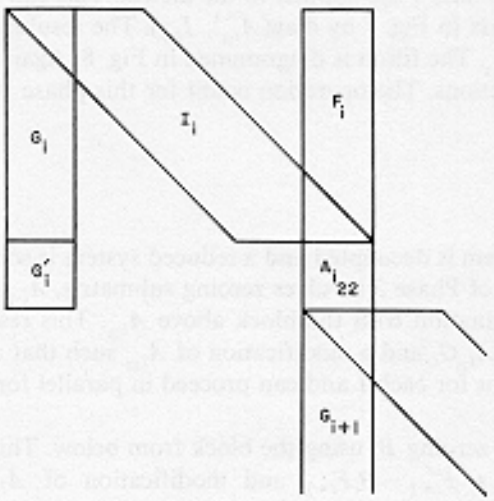

Fig. 9. Matrix after the first part of Phase 3.

where α falls in the interval 1–8 for block cyclic reduction and suitable architectures such as binary tree, shuffle-exchange, and boolean cube interconnected processors. For shared-memory architectures it is necessary that the bus/switch/storage bandwidth is proportional to $2P$ to realize $2 \log P$ communication start-ups. For two-way block Gaussian elimination the number of block row communications is P for a linear array and $2P$ for a shared-memory architecture. The parallel arithmetic complexity of block cyclic reduction is $(26m^3/3 + 8m^2r)(\log P - 2) + 16m^3/3 +$ *lower order terms* and for two-way Gaussian elimination $(8m^3/3 + 3m^2r)(P - 1) +$ *lower order terms*, r is the number of right-hand sides. The parallel arithmetic complexity of block cyclic reduction is always lower than that of two-way block Gaussian elimination. On most architectures the number of communication startups in sequence is also less for block cyclic reduction.

Detailed complexity estimates for both block Gaussian elimination and block cyclic reduction are given in [21] for a variety of architectures.

5.4. Phase 4

Given the solution to the block tridiagonal system, the complete solution to the original matrix can be found by a simple backsubstitution. Phase 4 can be done without communication of information from one section to another. This phase requires $4mkr$ operations, where r is the number of right-hand sides.

5.5. Complexity of concurrent elimination of different variables through partitioning

The parallel arithmetic complexity corresponding to the above partitioning strategy is given below:

Phase 1: $m(m+1)(N/P - m - (2m+1)/3)$ for a symmetric matrix,
$m^2(2(N/P - m) - 1) - (4m^2 - 1)m/3$ otherwise;
Phase 2: $m(2(N/P - m) - m - 1)(3m + 2r)$;
Phase 3: $2m(m+1)(m+r)$ + block trid. solve for a symmetric matrix,
$2m(m+1)(2m+r)$ + block trid. solve for an unsymmetric matrix;
Phase 4: $4m(N/P - m)$;

where

P = number of partitions,
N = order-of the matrix,
m = half bandwidth,
r = number of right-hand sides,
$P < N/m$.

For the above complexity estimates it is assumed that there is one processor per partition. The complexity of Phases 1, 2, and 4 has terms inversely proportional to the number of partitions, whereas the complexity of Phase 3 has terms that increase as a function of P. Hence, there exists a trade-off between time for the solution of the reduced system and the time required in the substructured elimination. The optimum number of partitions for linear arrays and shared-memory systems with a bandwidth independent of P is of order $\sqrt{N/m}$, whereas for boolean cubes, binary trees, shuffle-exchange networks and shared-memory systems with a bandwidth proportional to P the optimum number of partitions is of order N/m [30].

The parallel arithmetic complexities can be reduced further by performing operations on blocks concurrently, i.e., by parallelizing dense matrix factorization, solve and multiplication along the lines described for banded systems in Section 4. Such parallelization leads to increased communication complexity.

5.6. Algorithm properties

The partitioning strategy yields algorithms that have two significant properties:

(1) If the original matrix is symmetric, then the reduced block tridiagonal matrix is symmetric [21]. This is not the case with the partitioning employed by [8,28]. (Also if the original is diagonally dominant, so is the reduced system.)

(2) If the matrix is symmetric, then the condition number of partitioned matrices, $(A_{j_{11}})$, is never worse than that of the original matrix. This follows from the eigenvalue interacting properties of symmetric matrices.

The reduced system can be solved using block cyclic reduction. The advantage is that this phase can easily be parallelized.

The partitioning algorithm described here uses an elimination order that can be obtained through incomplete nested dissection [13]. A banded matrix with half bandwidth m corresponds to a graph in which each node is connected to all its preceding m nodes as well as to all its succeeding m nodes, except for the first and last sets of m nodes. Separators are of size m. The block matrices $A_{j_{22}}$ correspond to the edges between the nodes of a separator and the adjacent left and top triangular blocks to edges to one set of $n/P - m$ nodes. Choosing separators as bisectors recursively yields a complete binary tree as an elimination tree (Fig. 10).

The elimination order of the algorithm corresponds to the elimination of the leaf variables prior to the elimination of any variables of the internal nodes of the elimination tree. No particular order is implied for the elimination of variables in different leaf nodes, but the variables within a leaf node of the elimination tree are eliminated in order of increasing (or

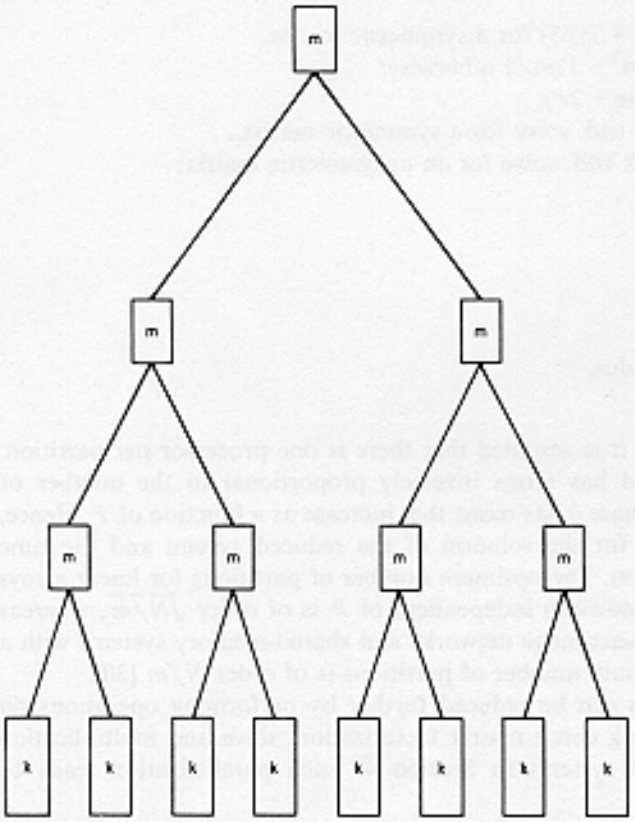

Fig. 10. Elimination tree generated by recursive bisection (incomplete nested dissection).

decreasing) labels. The elimination of the variables corresponding to the internal nodes from the leaves towards the root corresponds to the elimination order of block cyclic reduction, whereas an elimination in inorder yields block Gaussian elimination [27].

6. Complexity of the different strategies for concurrent elimination

The number of communication startups for the partitioning method described in Section 5 is potentially a factor of $\alpha m \log P/(2N\sqrt{P})$ less than that of the consecutive two-dimensional partitioning, which has the fewest startups of the algorithms exploiting the independence of operations for the elimination of a single variable. The data transfer may be a factor of $\alpha m \log P/(N\sqrt{P})$ times that of the consecutive two-dimensional partitioning. To fully realize this potential, the communication bandwidth of the architecture must be proportional to P, a configuration most easily accomplished in a network architecture.

We conclude that with respect to the communication complexity, algorithms exploiting the independence of operations for eliminating different variables offers substantial reductions compared to algorithms exploiting the independence of operations for eliminating single variables for most banded matrices.

The comparison with respect to the arithmetic complexity is not so favorable. The substructured elimination causes substantial fill-in. Indeed, the arithmetic complexity of Phases 1 and 2 alone is approximately 4 times that of the sequential algorithm for the general case (for $N/P - m$ equations), and 7 times for the symmetric case. Hence, the efficiency (speedup/(number of processors)) is limited, and the number of processing elements required to achieve any speedup is expected to be at least 5 to 10 if communication complexity and the time for the reduced system solve are included. A very modest speedup can be expected on current global storage with bus architectures because of the number of processors available. The situation for distributed storage architectures is more favorable in that they offer a larger number of processing elements.

Both kinds of concurrency can clearly be exploited simultaneously. The arithmetic complexity is reduced and the communication complexity increased. Exploiting both forms of concurrency may, however, represent a problem with respect to available compilers on some machines unless some other device is used to gain access to the parallel features [9] (such as the Alliant).

7. Experiments

The partitioning algorithm has been implemented on two shared-memory systems, the Sequent Balance 21000 (with 10 processors) and an Alliant FX/8 (with 8 vector-processors, with a vector register size of 32 elements) [1]. Results from the methods described in Section 4 are presented in the following two subsections.

7.1. Concurrent elimination of variables

7.1.1. Phase 1: Factorization

For Phase 1, a LINPACK routine is called. The peak performance of the band algorithm from LINPACK on the Alliant FX/8 is about 3 Mflops [18], which is also achieved for some phases of the algorithm. The vector features and the cache of the architecture have interesting effects on the performance, as indicated by the data for one processor as given in Table 3.

[1] For the case $m = 1$ (tridiagonal systems) also on the Intel iPSC/d7 and the Connection Machine [16].

Table 3
Execution time for factorization (in seconds) on Alliant FX/8, one processor used

m		$P=1$	$P=2$	$P=4$	$P=8$	$P=16$
2	$N=1000$	0.18	0.18	0.20	0.18	0.18
	$N=2000$	0.40	0.38	0.38	0.40	0.40
	$N=4000$	0.77	0.77	0.77	0.78	0.80
	$N=8000$	1.53	1.55	1.55	1.55	1.60
8	$N=1000$	0.42	0.42	0.40	0.40	0.37
	$N=2000$	0.83	0.83	0.85	0.83	0.83
	$N=4000$	1.67	1.68	1.70	1.70	1.75
	$N=8000$	3.37	3.40	3.43	3.47	3.63
16	$N=1000$	0.80	0.80	0.77	0.68	0.53
	$N=2000$	1.62	1.58	1.57	1.53	1.45
	$N=4000$	3.20	3.18	3.22	3.23	3.30
	$N=8000$	6.47	6.47	6.50	6.63	6.97
32	$N=1000$	1.72	1.63	1.40	0.98	0.33
	$N=2000$	3.47	3.42	3.17	2.70	1.90
	$N=4000$	7.05	6.85	6.77	6.30	5.38
	$N=8000$	14.55	14.68	14.65	14.92	15.55

For a small bandwidth the effect on the execution time due to an increased number of partitions is small. The total number of arithmetic operations is approximately $2m^2(N-(P-1)m)$. The overhead in managing loop iterations seems to dominate. But, for a half bandwidth of 32, the number of arithmetic operations is reduced significantly. The reduced number of arithmetic operations is part of the explanation for the reduced execution time for increasing P even if only one processor is used for the factorization. However, the reduction is much larger than predicted from the number of arithmetic operations alone as seen from Table 4. The running time is normalized to the time for $P=1$ for each N and m. The measured relative time is given first, the predicted $(1-(P-1)m/N)$ second.

Table 4
Relative execution time (measured:predicted) for factorization on Alliant FX/8, one processor used

m		$P=2$	$P=4$	$P=8$	$P=16$
2	$N=1000$	(1:1)	(1.11:1)	(1:0.98)	(1:0.97)
	$N=2000$	(0.95:1)	(0.95:1)	(1:0.99)	(1:0.98)
	$N=4000$	(1:1)	(1:1)	(1.01:1)	(1.04:0.99)
	$N=8000$	(1.01:1)	(1.01:1)	(1.01:1)	(1.05:1)
8	$N=1000$	(1:0.99)	(0.95:0.98)	(0.95:0.94)	(0.88:0.88)
	$N=2000$	(1:1)	(1.02:0.99)	(1:0.97)	(1:0.94)
	$N=4000$	(1.01:1)	(1.02:0.99)	(1.02:0.99)	(1.05:0.97)
	$N=8000$	(1.01:1)	(1.02:1)	(1.03:0.99)	(1.08:0.99)
16	$N=1000$	(1:0.98)	(0.96:0.95)	(0.85:0.89)	(0.66:0.76)
	$N=2000$	(0.98:0.99)	(0.97:0.98)	(0.94:0.94)	(0.90:0.88)
	$N=4000$	(0.99:1)	(1.01:0.99)	(1.01:0.97)	(1.03:0.94)
	$N=8000$	(1:1)	(1:0.99)	(1.02:0.99)	(1.08:0.97)
32	$N=1000$	(0.95:0.97)	(0.81:0.90)	(0.57:0.78)	(0.19:0.52)
	$N=2000$	(0.99:0.98)	(0.91:0.95)	(0.78:0.89)	(0.55:0.76)
	$N=4000$	(0.97:0.98)	(0.96:0.98)	(0.89:0.94)	(0.76:0.88)
	$N=8000$	(1.01:1)	(1.01:0.99)	(1.03:0.97)	(1.07:0.94)

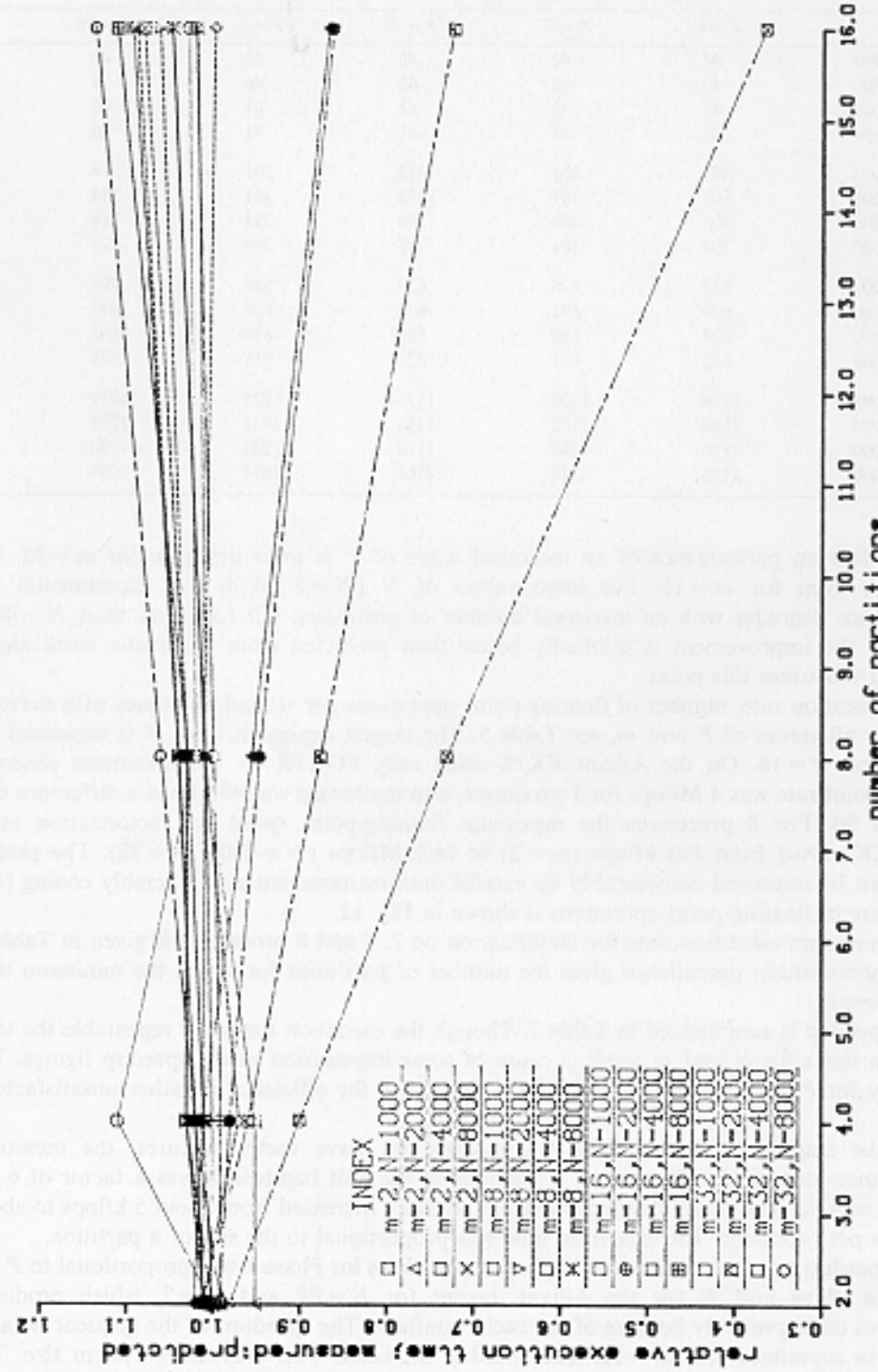

Fig. 11. Relative time for factorization on one processor as a function of P.

Table 5
Floating-point operations per second (kflops) for factorization on Alliant FX/8, one processor used

m		$P=1$	$P=2$	$P=4$	$P=8$	$P=16$
2	$N=1000$	44	44	40	44	43
	$N=2000$	40	42	42	40	39
	$N=4000$	42	42	42	41	40
	$N=8000$	42	41	41	41	40
8	$N=1000$	303	301	313	305	314
	$N=2000$	308	307	298	301	294
	$N=4000$	306	304	299	298	286
	$N=8000$	304	301	298	293	279
16	$N=1000$	633	626	635	686	794
	$N=2000$	629	641	638	639	643
	$N=4000$	638	640	629	620	593
	$N=8000$	632	631	627	611	575
32	$N=1000$	1164	1202	1335	1725	4039
	$N=2000$	1168	1172	1236	1731	1779
	$N=4000$	1156	1183	1184	1244	1390
	$N=8000$	1123	1110	1106	1074	1008

The effect on performance of an increased value of P is most dramatic for $m=32$, but significant even for $m=16$. For large values of N ($N=8000$ in our experiments) the performance degrades with an increased number of partitions, but for fewer than $N=4000$ equations the improvement is markedly better than predicted from arithmetic work alone. Figure 11 illustrates this point.

The execution rate, number of floating-point operations per second, increases with decreasing N for all values of P and m, see Table 5. The largest dependence on N is measured for $m=32$ and $P=16$. On the Alliant FX/8 using only FORTRAN the maximum observed floating-point rate was 4 Mflops for 1 processor, and the lowest was 40 kflops a difference of a factor of 90. For 8 processors the maximum floating-point speed for factorization using LINPACK varied from 193 kflops ($m=2$) to 16.1 Mflops ($N=2000$, $m=32$). The performance can be improved considerably by careful data management and assembly coding [18].

The rate of floating-point operations is shown in Fig. 12.

The minimum execution time for factorization on 2, 4 and 8 processors is given in Table 6. The numbers within parentheses gives the number of partitions for which the minimum time was achieved.

The speedup is summarized in Table 7. Though the execution times are repeatable the total execution times for N and m small is cause of some imprecision in the speedup figures. The efficiency for $P=4$ is in the range 75–90%. For $P=8$ the efficiency is rather unsatisfactory, 55–85%.

For the Sequent Balance 21000, which does not have vector features, the measured performance increase for Phase 1 as a function of the half bandwidth was a factor of 6 for $m=2$ to $m=32$. The actual floating-point performance increased from about 5 kflops to about 30 kflops per processor. The execution time was proportional to the size of a partition.

The speedup as a function of the number of partitions for Phase 1 was proportional to P for all values of m and N for the Alliant, except for $N=8k$ and $m=2$, which produced anomalous data, probably because of the cache conflicts. The speedup for the Sequent Balance 21000 was superlinear for $m=32$. The speedup increased with decreasing system size. The speedup for $m=2$ was slightly sublinear for $N=2k$ and strongly sublinear for smaller systems. We cannot account for this phenomenon.

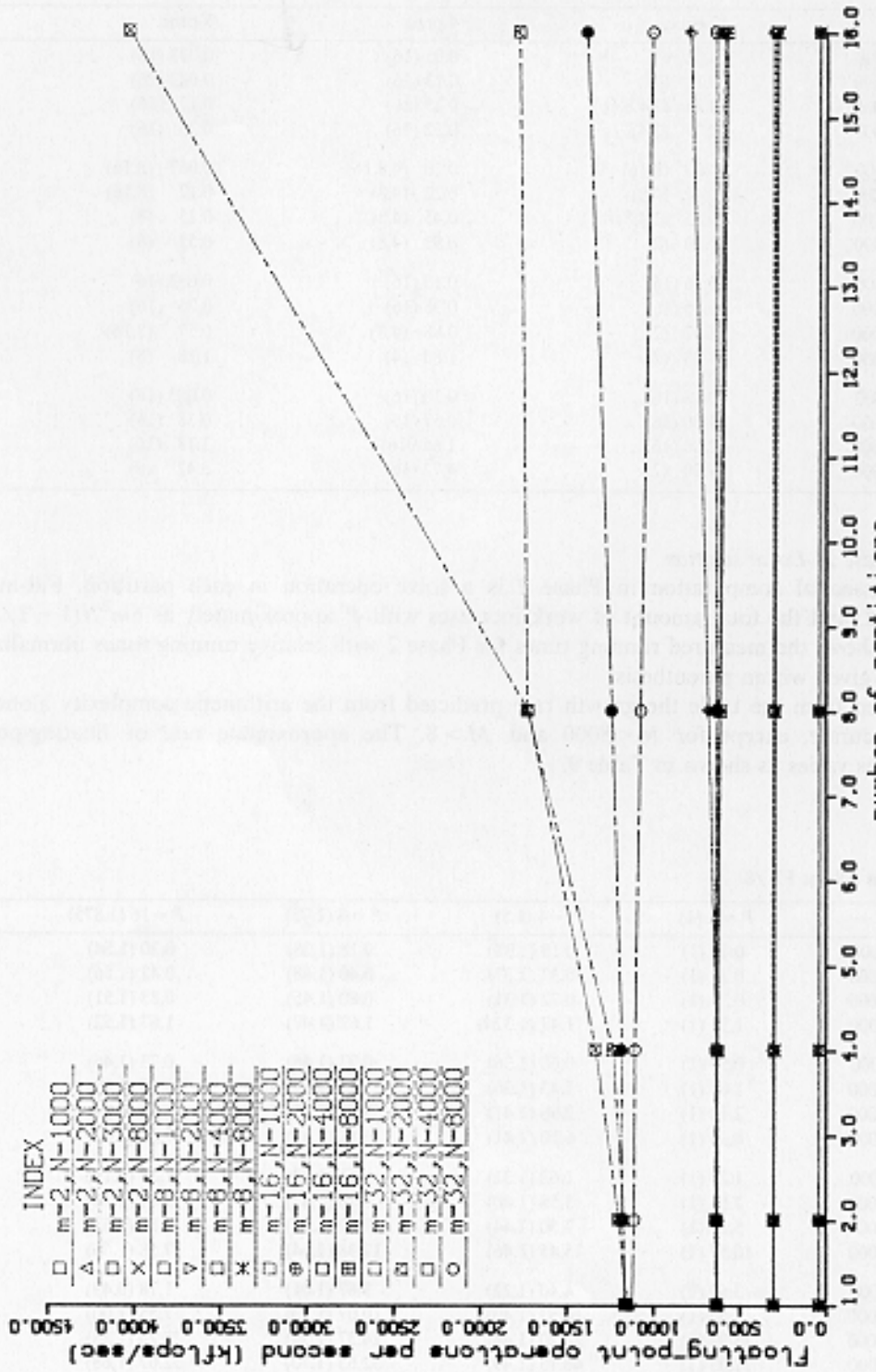

Fig. 12. Achieved floating-point speed on one processor during factorization.

Table 6
Execution time for factorization (in seconds) on Alliant FX/8

m		2 proc.	4 proc.	8 proc.
2	$N = 1000$	0.08 (4)	0.05 (16)	0.033 (16)
	$N = 2000$	0.18 (4)	0.13 (16)	0.083 (16)
	$N = 4000$	0.38 (2,4,8,16)	0.25 (16)	0.17 (16)
	$N = 8000$	0.77 (2,4)	0.52 (16)	0.33 (16)
8	$N = 1000$	0.20 (8.16)	0.10 (4,8,16)	0.067 (8.16)
	$N = 2000$	0.42 (4,8)	0.22 (4,8)	0.12 (8,16)
	$N = 4000$	0.85 (2,4,8)	0.45 (4,8)	0.25 (8)
	$N = 8000$	1.70 (2)	0.92 (4,8)	0.52 (8)
16	$N = 1000$	0.26 (16)	0.13 (16)	0.083 (16)
	$N = 2000$	0.75 (16)	0.38 (16)	0.25 (16)
	$N = 4000$	1.62 (2)	0.88 (4,8)	0.57 (8,16)
	$N = 8000$	3.28 (2)	1.80 (4)	1.18 (8)
32	$N = 1000$	0.18 (16)	0.10 (16)	0.083 (16)
	$N = 2000$	1.00 (16)	0.57 (16)	0.38 (16)
	$N = 4000$	2.80 (16)	1.65 916)	1.18 (16)
	$N = 8000$	7.80 (2)	4.72 (16)	3.47 (8)

7.1.2. Phase 2: Local solution

The essential computation in Phase 2 is a solve operation in each partition. Fill-in is generated, and the total amount of work increases with P approximately as $6m^2N(1 - 1/P)$. Table 8 shows the measured running times for Phase 2 with relative running times normalized to $P = 2$ given within parenthesis.

As seen from the table the growth rate predicted from the arithmetic complexity alone is fairly accurate, except for $N < 8000$ and $M > 8$. The approximate rate of floating-point operations varies as shown in Table 9.

Table 7
Speedup on Alliant FX/8

m		$P = 2$ (1)	$P = 4$ (1.5)	$P = 8$ (1.75)	$P = 16$ (1.875)
2	$N = 1000$	0.13 (1)	0.18 (1.38)	0.18 (1.38)	0.20 (1.54)
	$N = 2000$	0.27 (1)	0.37 (1.37)	0.40 (1.48)	0.42 (1.56)
	$N = 4000$	0.55 (1)	0.72 (1.31)	0.80 (1.45)	0.83 (1.51)
	$N = 8000$	1.10 (1)	1.47 (1.324)	1.62 (1.47)	1.67 (1.52)
8	$N = 1000$	0.50 (1)	0.68 (1.36)	0.73 (1.46)	0.73 (1.46)
	$N = 2000$	1.05 (1)	1.43 (1.36)	1.57 (1.50)	1.58 (1.50)
	$N = 4000$	2.12 (1)	2.98 (1.41)	3.28 (1.55)	3.33 (1.57)
	$N = 8000$	4.33 (1)	6.10 (1.41)	6.87 (1.59)	7.02 (1.62)
16	$N = 1000$	1.23 (1)	1.62 (1.32)	1.63 (1.33)	1.37 (1.11)
	$N = 2000$	2.55 (1)	3.58 (1.40)	3.70 (1.45)	3.47 (1.36)
	$N = 4000$	5.20 (1)	7.50 (1.44)	8.22 (1.58)	7.88 (1.52)
	$N = 8000$	10.57 (1)	15.45 (1.46)	17.38 (1.64)	17.58 (1.66)
32	$N = 1000$	3.62 (1)	4.40 (1.22)	3.67 (1.01)	1.78 (0.49)
	$N = 2000$	7.55 (1)	10.57 (1.40)	10.07 (1.33)	7.55 (1.00)
	$N = 4000$	15.37 (1)	22.20 (1.44)	24.37 (1.59)	21.28 (1.38)
	$N = 8000$	31.03 (1)	46.15 (1.49)	52.65 (1.70)	52.07 (1.68)

Table 8
Measured running times (in seconds) for solve on Alliant FX/8, one processor used

m		$P=2$ (1)	$P=4$ (1.5)	$P=8$ (1.75)	$P=16$ (1.875)
2	$N=1000$	0.13 (1)	0.18 (1.38)	0.18 (1.38)	0.20 (1.54)
	$N=2000$	0.27 (1)	0.37 (1.37)	0.40 (1.48)	0.42 (1.56)
	$N=4000$	0.55 (1)	0.72 (1.31)	0.80 (1.45)	0.83 (1.51)
	$N=8000$	1.10 (1)	1.47 (1.324)	1.62 (1.47)	1.67 (1.52)
8	$N=1000$	0.50 (1)	0.68 (1.36)	0.73 (1.46)	0.73 (1.46)
	$N=2000$	1.05 (1)	1.43 (1.36)	1.57 (1.50)	1.58 (1.50)
	$N=4000$	2.12 (1)	2.98 (1.41)	3.28 (1.55)	3.33 (1.57)
	$N=8000$	4.33 (1)	6.10 (1.41)	6.87 (1.59)	7.02 (1.62)
16	$N=1000$	1.23 (1)	1.62 (1.32)	1.63 (1.33)	1.37 (1.11)
	$N=2000$	2.55 (1)	3.58 (1.40)	3.70 (1.45)	3.47 (1.36)
	$N=4000$	5.20 (1)	7.50 (1.44)	8.22 (1.58)	7.88 (1.52)
	$N=8000$	10.57 (1)	15.45 (1.46)	17.38 (1.64)	17.58 (1.66)
32	$N=1000$	3.62 (1)	4.40 (1.22)	3.67 (1.01)	1.78 (0.49)
	$N=2000$	7.55 (1)	10.57 (1.40)	10.07 (1.33)	7.55 (1.00)
	$N=4000$	15.37 (1)	22.20 (1.44)	24.37 (1.59)	21.28 (1.38)
	$N=8000$	31.03 (1)	46.15 (1.49)	52.65 (1.70)	52.07 (1.68)

The floating-point rate generally increases with m and P, and decreases as a function of N. For $m=2$ the rate is about 3 times higher than the factorization rate, approximately 40% higher for $m=8$, about the same for $m=16$, and only 65% of the rate for factorization for $m=32$. The rate varies by a factor of 8 (compared to a variation by a factor 90 for the factorization). The floating-point rate is shown in Fig. 13.

The speedup for the solve routine if the number of partitions matches the number of processing elements is expected to be 1.5, 3, 7, and 15 for $P=2$, $P=4$, $P=8$, and $P=16$, respectively, compared to the one-processor case (with the corresponding value of P). Note that the total work increases with P. The speedup with P being equal to the number of

Table 9
Floating-point operations per second (kflops) for solve on Alliant FX/8, one processor used

m		$P=2$	$P=4$	$P=8$	$P=16$
2	$N=1000$	122	131	151	142
	$N=2000$	118	129	138	139
	$N=4000$	116	133	139	143
	$N=8000$	116	130	138	143
8	$N=1000$	406	436	449	427
	$N=2000$	391	425	441	444
	$N=4000$	390	416	433	445
	$N=8000$	383	407	419	433
16	$N=1000$	619	668	691	666
	$N=2000$	612	638	683	695
	$N=4000$	608	624	648	687
	$N=8000$	602	614	629	649
32	$N=1000$	782	862	915	740
	$N=2000$	790	804	879	953
	$N=4000$	796	806	813	891
	$N=8000$	799	796	793	816

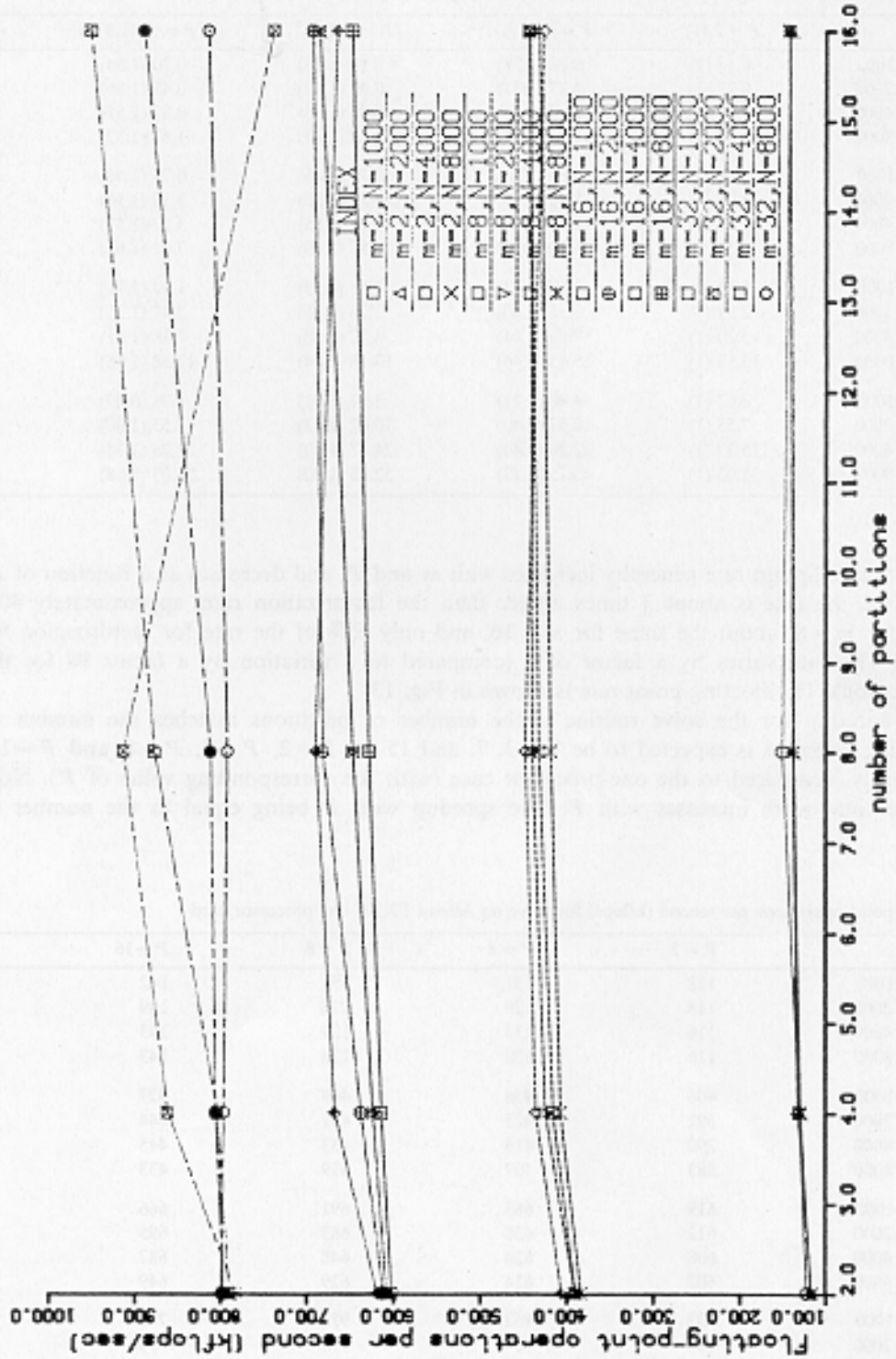

Fig. 13. Floating-point rate of forward and backward solve on one processor.

Table 10
Execution times (in seconds) for solve on Alliant FX/8

m		2 proc.	4 proc.	8 proc.
2	$N=1000$	0.10 (16)	0.067 (16)	0.033 (16)
	$N=2000$	0.23 (8.16)	0.15 (16)	0.10 (16)
	$N=4000$	0.47 (16)	0.30 (16)	0.20 (16)
	$N=8000$	0.93 (16)	0.62 (16)	0.40 (16)
8	$N=1000$	0.40 (16)	0.20 (16)	0.12 916)
	$N=2000$	0.88 (16)	0.47 (16)	0.30 (8.16)
	$N=4000$	1.87 (16)	1.02 (16)	0.62 (8.16)
	$N=8000$	3.97 (16)	2.13 (16)	1.27 (8.16)
16	$N=1000$	0.78 (16)	0.42 (16)	0.25 (16)
	$N=2000$	1.92 (16)	1.07 (16)	0.70 (16)
	$N=4000$	4.47 (16)	2.47 (16)	1.58 (16)
	$N=8000$	9.73 (16)	5.27 (16)	3.37 (16)
32	$N=1000$	0.98 (16)	0.53 (16)	0.32 (16)
	$N=2000$	4.27 (16)	2.45 (16)	1.65 (16)
	$N=4000$	12.25 (16)	6.87 (16)	4.58 (16)
	$N=8000$	28.35 (16)	15.52 (16)	10.45 (16)

partitions is $P/(3m)$ compared to the sequential algorithm, which amounts to a slow-down for the cases reported here. The figures in parenthesis in Table 10 for the execution time for the different number of processors give the numer of partitions for which the minimum execution time was achieved. The fact that $P=16$ seemingly always is the best of the considered values is in part due to the fact that the compiler failed to properly parallelized one loop. The speedup is summarized in Table 11.

The speedup for 2 processors is 80–100% of the theoretical value,for 4 processors 60–100%, and for 8 processors 40–80%. Part of the reason for the discrepancy between the expected speedup and the measured speedup for the solve phase is due to the characteristics of the

Table 11
Speedup of solve on Alliant FX/8

m		2 proc.	4 proc.	8 proc.
2	$N=1000$	1.30	1.94	3.93
	$N=2000$	1.17	1.80	2.70
	$N=4000$	1.17	1.83	2.75
	$N=8000$	1.18	1.77	2.75
8	$N=1000$	1.25	2.50	4.17
	$N=2000$	1.19	2.23	3.50
	$N=4000$	1.13	2.08	3.42
	$N=8000$	1.09	2.03	3.41
16	$N=1000$	1.58	2.93	4.92
	$N=2000$	1.33	2.38	3.64
	$N=4000$	1.16	2.11	3.29
	$N=8000$	1.09	2.01	3.14
32	$N=1000$	1.82	3.36	5.56
	$N=2000$	1.77	3.08	4.58
	$N=4000$	1.25	2.24	3.36
	$N=8000$	1.09	2.00	2.97

compiler for the Alliant FX/8. It parallelizes one loop level (outermost unless instructed otherwise). It fails to parallelize the solve code for two partitions, since the code handles the fill-in corresponding to a distinct set of solution variables in each iteration. Writing the code such that each loop iteration contains only the operation for one partition forces the introduction either of conditionals to handle the first and last partition which are special cases, or sequential code outside the loop for these partitions. In either case the compiler either fails completely to parallelize, or has to be 'tricked' into parallelizing the code properly. For sufficiently large values of P this problem should be of minor influence.

A second reason for the discrepancy is due to data transfer rates between main memory and cache memory, and between cache memory to the vector registers. The data transfer rate from the cache memory to the processor's registers is 376 Mbytes/s and from main memory to cache memory is 188 Mbytes/s. The cache itself can hold 128 kbytes of data (16k words). The machine architecture is balanced between the execution rate of the 8 processors and the transfer rate from cache, however if the data request is not in cache and a 'real' memory reference is required there is an imbalance. When 8 processors are active, each requesting one operand of a vector operation from memory and the operand is in the cache, the cache can supply the processors with data at 376 Mbytes/s to achieve the peak performance of about 5.9 Mflops per processor (46.4 Mbytes/s/processor). When a data reference is not in the cache memory and thus the requested data must come from main memory, the transfer rate is cut in half to 188 Mbytes/s. With four or less processors active the rate from memory can keep up with the execution rate from the active processors, however as more processors are active and request data from memory the mismatch between the transfer rates and execution speed becomes apparent and the execution rate drops.

To overcome this limitation the algorithm must be reorganized to get more reuse of the data which in this case means matrix-matrix operations. The net effect for our program is to have poor performance relative to the speedup measurements.

The efficiency of the partitioning method is critically dependent on the time for the solve routine compared to the factorization routine for small values of P. Most of the time spent in the solve routine is due to fill-in, that is, the price paid for the partitioning strategy (or the

Table 12
Solve/factorization (time in seconds) Alliant FX/8 one processor used

m		$P = 2$ (1.5)	$P = 4$ (2.25)	$P = 8$ (2.625)
2	$N = 1000$	1.25	1.34	1.00
	$N = 2000$	1.28	1.15	1.20
	$N = 4000$	1.24	1.20	1.18
	$N = 8000$	1.21	1.19	1.21
8	$N = 1000$	2.00	2.00	1.79
	$N = 2000$	2.10	2.14	2.50
	$N = 4000$	2.20	2.27	2.487
	$N = 8000$	2.34	2.32	2.44
16	$N = 1000$	3.00	3.23	3.01
	$N = 2000$	2.56	2.82	2.80
	$N = 4000$	2.76	2.81	2.77
	$N = 8000$	2.97	2.93	2.86
32	$N = 1000$	5.44	5.30	3.86
	$N = 2000$	4.27	4.30	4.34
	$N = 4000$	4.38	4.16	3.88
	$N = 8000$	3.63	3.29	3.01

Table 13
Phase 2/phase 1 on Sequent Balance 21000

m	$N = 2048$	$N = 1024$	$N = 512$	$N = 256$
2	2.78	2.89	2.5	2.9
5	4.13	4.3	4.0	4.1
16	4.46	4.4	4.4	4.7
32	4.61	4.6	4.6	–

nested dissection elimination order). For all partitions but the first and last the ratio of the arithmetic complexity for the solve to that of factorization is approximately 3. For one processor the ratio is $3(1 - 1/P)$.

As can be seen from Table 12 the relative solve time is lower than expected for $m = 2$, higher than expected for $m = 8$ and $P = 2$, and slightly lower than expected for $m = 8$, $P > 2$. For $m > 8$ the relative solve time is higher than expected, and significantly so for $m = 32$. Consequently, the LINPACK-based parallel algorithm performing concurrent elimination of variables is less time-consuming than predicted for small bandwidths. With the exception of the $N = 8k$ case Phase 2 on the Alliant requires less time than predicted, if overhead is ignored. Note that the ratio is relatively independent of N, as expected, and also (approximately) independent of m. It is also interesting to note that the ratio is approximately the same for $m = 2$ and $m = 32$, but is higher for intermediate values (by 30–50%). For the Sequent Balance 21000 the ratio is generally higher than expected, and increases as a function of m, with a relatively small increase for $m > 8$, see Table 13.

7.1.3. Phase 3: Solution of the reduced block tridiagonal system

For block Gaussian elimination the expected solution time is proportional to P and m^3. The measured performance is approximately linear in P, but doubling the bandwidth only quadruples the solution time on the Alliant, see Table 14. Being inherently sequential, with the exception for the possibility to perform two-way elimination, the execution time should be independent of the number of processing elements, since no parallelization of block matrix (LINPACK) operations is performed. The measurements occur with the predicted behavior.

The block cyclic reduction solver requires the same number of arithmetic operations for $P = 2$ and $P = 4$. The total number of arithmetic operations for $P = 8$ is higher. The block cyclic reduction solver requires approximately 1.26 times as many operations as the block Gaussian elimination solver, but the parallel arithmetic complexity is lower. However, the Alliant compiler did not manage to parallelize our cyclic reduction code. The running time for $P = 4$ was approximately 1.15 times that of the Gaussian elimination code, and for $P = 8$ it was approximately 1.52 for 1 processor, 1.42 for 2 processors, and 1.40 for 4 or more processors.

For the Sequent Balance 21000 the dependence is approximately linear in P and approximately cubic in m, see Table 15.

Table 14
Time for block Gaussian elimination on Alliant FX/8

m	$P = 2$	$P = 4$	$P = 8$	$P = 16$
2	0	0	0	0
8	0	0.03	0.06	0.13
16	0.01	0.08	0.22	0.50
32	0.03	0.3	0.90	2.00

Table 15
Time for block Gaussian elimination on Sequent Balance 21000

m	$P=4$	$P=8$
2	0.005	0.075
8	0.583	1.25
16	3.60	7.65
32	25.25	53.63

7.1.4. Phase 4: Back substitution

The time for Phase 4 is small relative to the time for the other phases and offers no particular insight.

7.1.5. Total time for the partitioning method

The total time required to solve the different banded systems using block Gaussian elimination for the reduced system is given in Table 16.

Our implementation of the partitioning strategy on the Alliant FX/8 yields a speedup for $P > 1$, $m = 2$, for $P \geq 4$, $m = 8$, and for $P \geq 8$, $m = 16$. For half bandwidth 32 there is a slight speedup for $N \geq 4000$, and a slight slowdown for smaller systems. The total time for the partitioning method on the Alliant is shown as a function of P with N as a parameter in Fig. 14 for $m = 2$ and $m = 32$.

7.2. One-dimensional partitioning

As an alternative to the concurrent elimination of different variables, a one-dimensional partitioning can be used for the concurrent elimination of a single variable. Table 17 gives the result from performing column operations in parallel.

For a small bandwidth the parallelism for the column partitioning is very limited, and as seen for the case $m = 2$ the execution time does not decrease beyond the time for 2 processors.

Table 16
Total time (in seconds) to solve using block Gaussian elimination on Alliant FX/8

m		$P=1$	$P=2$	$P=4$	$P=8$
2	$N=1000$	0.27	0.25	0.18	0.12
	$N=2000$	0.52	0.48	0.32	0.23
	$N=4000$	1.00	0.92	0.60	0.42
	$N=8000$	1.98	1.83	1.20	0.80
8	$N=1000$	0.50	0.75	0.42	0.30
	$N=2000$	1.00	1.47	0.82	0.52
	$N=4000$	2.00	2.97	1.62	0.98
	$N=8000$	4.02	5.98	3.22	1.93
16	$N=1000$	0.93	1.68	0.98	0.75
	$N=2000$	1.87	3.45	1.92	1.35
	$N=4000$	3.67	6.98	3.81	2.55
	$N=8000$	7.38	13.98	7.55	4.97
32	$N=1000$	1.92	4.25	2.67	2.12
	$N=2000$	3.87	8.53	5.35	3.97
	$N=4000$	7.82	18.43	10.73	7.63
	$N=8000$	16.08	39.55	22.03	15.55

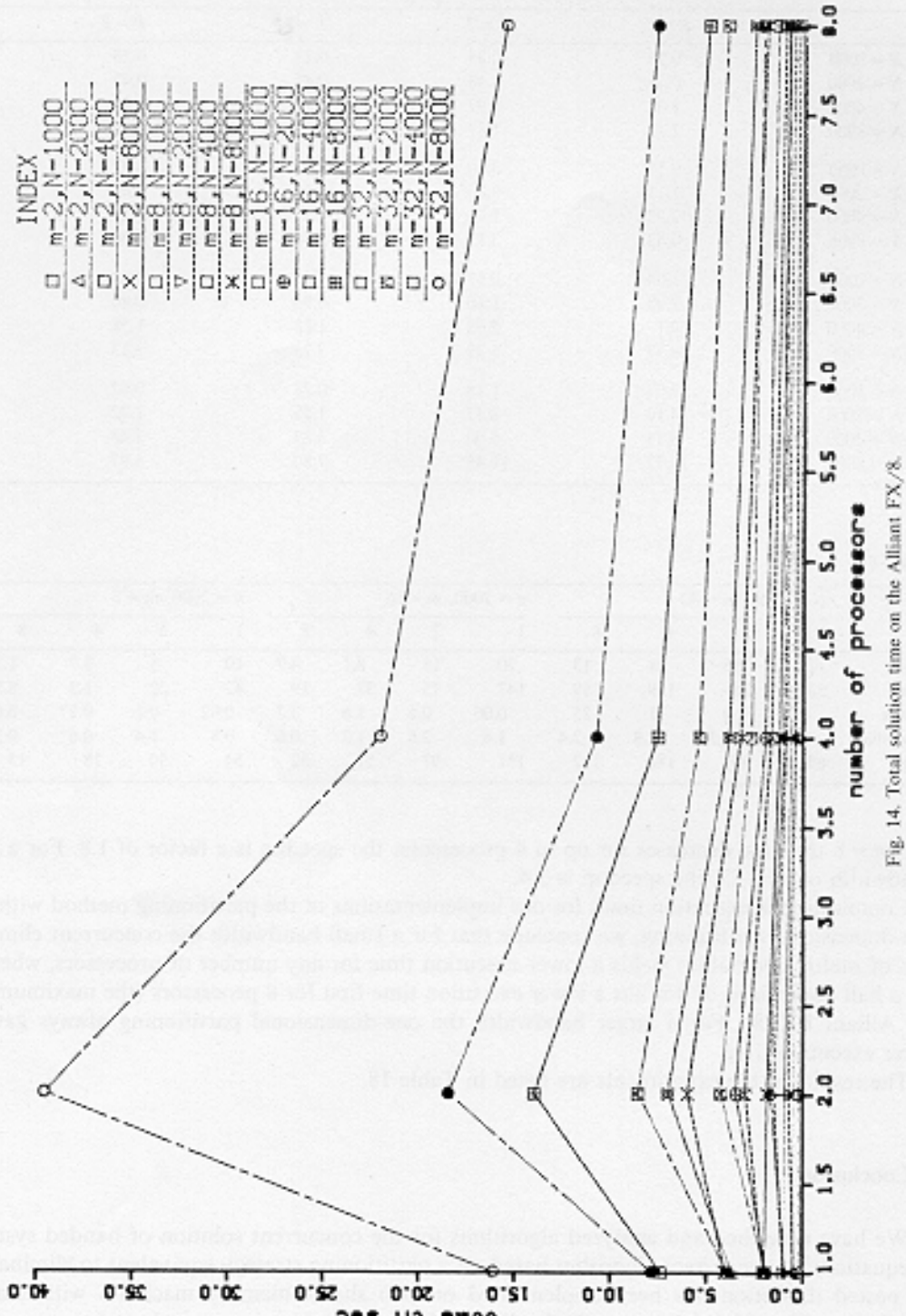

Fig. 14. Total solution time on the Alliant FX/8.

Table 17
One-dimensional partitioning (time in seconds) on Alliant FX/8

m		$P=1$	$P=2$	$P=4$	$P=8$
2	$N=1000$	0.28	0.25	0.23	0.23
	$N=2000$	0.55	0.48	0.45	0.43
	$N=4000$	1.07	0.97	0.85	0.87
	$N=8000$	2.12	1.87	1.70	1.70
8	$N=1000$	0.58	0.40	0.31	0.32
	$N=2000$	1.13	0.80	0.63	0.60
	$N=4000$	2.23	1.55	1.25	1.15
	$N=8000$	4.42	3.12	2.48	2.30
16	$N=1000$	1.08	0.67	0.48	0.42
	$N=2000$	2.08	1.30	0.93	0.80
	$N=4000$	4.17	2.62	1.88	1.58
	$N=8000$	8.37	5.22	3.73	3.13
32	$N=1000$	2.07	1.18	0.78	0.62
	$N=2000$	4.10	2.35	1.55	1.22
	$N=4000$	8.75	5.00	3.23	2.48
	$N=8000$	19.77	11.45	7.80	5.97

Table 18

	$N=2000$, $m=32$				$n=2000$, $m=16$				$n=2000$, $m=8$			
	1	2	4	8	1	2	4	8	1	2	4	8
Factor	113	55	28	13	30	15	8.1	4.7	10	5	2.7	1.3
Solve	535	269	129	59	147	75	37	19	42	22	1.3	8.5
Reduce	0.5	4	11	25	0.06	0.6	1.6	3.7	0.02	0.1	0.27	0.6
Backsolve	3	5.3	2.8	2.4	1.6	2.6	1.2	0.6	0.8	1.4	0.6	0.3
Total	658	343	180	112	181	97	51	32	54	30	18	13

For $m=8$ the time decreases for up to 4 processors; the speedup is a factor of 1.8. For a half bandwidth of $m=32$, the speedup is 3.4.

Comparing the execution times for our implementations of the partitioning method with the one-dimensional partitioning, we conclude that for a small bandwidth the concurrent elimination of multiple variables yields a lower execution time for any number of processors, whereas for a half bandwidth of 8 it has a lower execution time first for 8 processors (the maximum on the Alliant FX/8). For a larger bandwidth the one-dimensional partitioning always gave a lower execution time.

The results of the experiments are listed in Table 18.

8. Conclusion

We have described and analyzed algorithms for the concurrent solution of banded systems of equations. A concurrent algorithm based on a partitioning strategy equivalent to elimination by nested dissection has been implemented on two shared-memory machines with limited parallelism. The partitioning method offers a low communication complexity compared to concurrent elimination of single variables, and has numerical advantages over some previously implemented partitioning methods.

The arithmetic complexity of the parallel algorithm is higher than that of standard direct solvers for band matrix problems. At least 5 processors are needed for the parallel arithmetic complexity to be lower than that of standard Gaussian elimination.

Because the computation of the fill-in is relatively more efficient than the factorization for small bandwidths the measured running time shows that our LINPACK-based implementation of the partitioning algorithm on the Alliant FX/8 has a lower execution time than the sequential algorithm (using LINPACK) for any number of processors. However, for bandwidths approaching the length of the vector registers, the factorization routine uses the architecture more efficiently than the solve routine, and the computation of the fill-in is relatively more expensive than predicted, moving the break-even point for the partitioning method to a higher number of processors than predicted. Indeed, for $m = 32$ the break-even point occurs at 8 processors in our implementation. Part of the reason for the higher than predicted break-even point is the fact that the compiler failed to parallelize some computations that can be done in parallel. Another reason is the mismatch between the transfer rate to the processors from the cache memory and the main memory. If several loop levels could be parallelized, then a higher processor utilization and lower execution time are likely. (The current Alliant compiler allows parallelization only of one loop level.)

The one-dimensional partitioning yields a speedup that is insignificant for a half bandwidth of $m = 2$, but is 3.4 for 8 processors and a half bandwidth of $m = 32$. Consequently, the concurrent elimination of multiple variables yields a lower execution time for any number of processors for a half bandwidth of 2, for 8 processors at a half bandwidth of 8. Since the Alliant is limited to 8 processors we were unable to establish a break-even point for larger bandwidths.

Acknowledgment

We would like to thank Tom Hewitt from Cray Research for his assistance with the CRAY data.

References

[1] C. Ashcroft, Parallel reduction methods for the solution of banded systems of equations.

[2] S.N. Bhatt and I.C.F. Ipsen, How to embed trees in hypercubes, Department of Computer Science, Yale University, Report YALEU/CSD/RR-443, 1985.

[3] W. Crowther, J. Goodhue, E. Starr, R. Thomas, W. Milliken and T. Blackadar, Performance measurements on a 128-node butterfly parallel processor, *Proc. 1985 IEEE International Conference on Parallel Processing* (1985) 531–540.

[4] S.R. Desphande and R.M. Jenevin, Scalability of a binary tree on a hypercube, University of Texas at Austin, Report TR-86-01, 1986.

[5] J. Dongarra, J. Bunch, C. Moler and G. Stewart, *LINPACK Users' Guide* (SIAM, Philadelphia, 1976).

[6] J. Dongarra and S.C. Eisenstat, Squeezing the most out of an algorithm in Cray Fortran, *ACM Trans. Math. Software* **10** (3) (1984) 221–230.

[7] J.J. Dongarra, F.G. Gustavson and A. Karp, Implementing linear algebra algorithms for dense matrices on a vector pipeline machine, *SIAM Rev.* **26** (1) (1984) 91–112.

[8] J.J. Dongarra and A. Sameh, On some parallel banded system solvers, *Parallel Comput.* **1** (3) (1984) 223–235.

[9] J. Dongarra and D. Sorensen, On environment for implementing explicite parallel processing in Fortran, ANL MCS, TM 79, 1986.

[10] D. Evans and M. Hatzopoulos, The solution of certain banded systems of linear equations using the folding algorithm, *Computer J.* **19** (1976) 184–187.

[11] M.J. Flynn, Very high-speed computing systems, *Proc. IEEE* **12** (1966) 1901–1909.

[12] M.W. Gentleman, Implementing nested dissection, Department of Computer Science, University of Waterloo, Research Report CS-82-03, 1982.

[13] A. George, Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.* **10** (1973) 345–363.
[14] A. George, M. Heath, J. Liu and E. Ng, Sparse Cholesky factorization on a local-memory multiprocessor, *SIAM J. Sci. Statist. Comput.*, to appear.
[15] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph and M. Snir, The NYU Ultracomputer—Designing an MIMD shared memory parallel computer, *IEEE Trans. Comput.* **32** (1983) 175–189.
[16] W.D. Hillis, *The Connection Machine* (MIT Press, Cambridge, MA, 1985).
[17] C.-T. Ho and S.L. Johnsson, Tree embeddings and optimal routine in hypercubes, Yale University, Department of Computer Science, Report in preparation.
[18] W. Jalby and U. Meier, Optimizing matrix operations on a parallel multiprocessor with a memory hierarchy, Center for Supercomputer Research and Development, University of Illinois, 1986.
[19] S.L. Johnsson, Gaussian elimination on sparse matrices and concurrency, Caltech Computer Science Department, 4087, TR:80, 1980.
[20] S.I. Johnsson, Odd-even cyclic reduction on ensemble architectures and the solution of tridiagonal systems of equations, Department of Computer Science, Yale University, Report YALEU/CSD/RR-339, 1984.
[21] S.L. Johnsson, Fast banded systems solvers for ensemble architectures, Department of Computer Science, Yale University, Report YALEU/CSD/RR-379, 1985
[22] S.L. Johnsson, Dense matrix operations on a torus and a boolean cube, *Proc. National Computer Conference* (AFIPS, Chicago 1985).
[23] S.L. Johnsson, Communication efficient basic linear algebra computations on hypercube architectures, Department of Computer Science, Yale University, Report YALEU/CSD/RR-361, 1985.
[24] S.L. Johnsson, Band matrix systems solvers on ensemble architectures, Yale University, Report YALEU/CSD/RR-388, 1985.
[25] S.L. Johnsson, Solving narrow banded systems on ensemble architecture, *ACM Trans. Math. Software* **11** (1985).
[26] S.L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Statist. Comput.* (1986).
[27] S.L. Johnsson, Data permutations and basic linear algebra computations on ensemble architectures, Department of Computer Science, Yale University, Report YALEU/CSD/RR-367, 1985.
[28] D. Lawrie and A. Sameh, The computation and communication complexity of parallel banded system solves, *ACM Trans. Math. Software* (1985).
[29] C. Lawson, R. Hanson, D. Kincaid and F. Krogh, Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software* (1979) 308–323.
[30] J.W.H. Liu, Computational models and task scheduling for parallel sparse Cholesky factorization, Department of Computer Science, York University, Downsview, Ontario, Technical Report CS-85-01, 1985.
[31] O.A. McBryan and E.F. Van de Velde, Hypercube algorithms and implementations, Courant Institute of Mathematical Sciences, New York University, 1985.
[32] U. Meier, A parallel partition method for solving banded systems of linear equations, *Parallel Comput.* **2** (1985) 33–45.
[33] G.F. Pfister, W.C. Brantley, D.A. George, S.I. Harvey, W.J. Kleinfelder, K.P. McAuliffe E.A. Melton, V.A. Norton and J. Weiss, The IBM research parallel processor prototype (RP3): Introduction and architecture, *Proc. 1985 IEEE International Conference on Parallel Processing* (1985) 764–771.
[34] R. Read and D.J. Rose, *Graph Theory and Computations* (Academic Press, New York, 1973) 183–217.
[35] E.M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms* (Prentice-Hall, Englewood Cliffs, NJ, 1977).
[36] Y. Saad and M.H. Schultz, Data communication in hypercubes, Department of Computer Science, Yale University, Report YALEU/DCS/RR-428, 1985.
[37] J.T. Schwartz, Ultracomputers, *ACM Trans. Programming Languages Systems* **2** (1980) 484–521.
[38] C.L. Seitz, The cosmic cube, *Comm. ACM* **28** (1) (1985) 22–33.
[39] B.J. Smith, Architecture and applications of the HEP multiprocessor computer system, *Real-Time Signal Processing IV, Proc. of SPIE* (1981) 241–248.
[40] D.C. Sorensen, Buffering for vector performance on a pipelined MIMD machine, *Parallel Comput.* **1** (1984) 143–164.
[41] J. Wilkinson, Private communication, 1976.
[42] O. Wing and J.W. Huang, A computational model of parallel solution of linear equations, *IEEE Trans. Comput.* **29** (1980) 632–638.
[43] P.H. Worley and R. Schreiber, Nested dissection on a mesh-connected processor array, Stanford University, Center for Large Scale Scientific Computation, CLaSSiC-85-08, 1985.