SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 824-842, July 1991

PARALLEL ALGORITHMS FOR BANDED LINEAR SYSTEMS*

STEPHEN J. WRIGHT†

Abstract. A partitioned Gaussian elimination algorithm with partial pivoting which is suitable for multiprocessors with small to moderate numbers of processing elements is described. It is only assumed that the system is nonsingular; hence the submatrices in the chosen partitioning may be rank-deficient and this makes the algorithm more complex than those which have been proposed for diagonally dominant and symmetric positive-definite systems. Operation counts and stability are examined. Some numerical results obtained on Alliant FX/8 and Sequent Balance multiprocessors are presented.

Key words. banded matrices, parallel processors

AMS(MOS) subject classifications. 65F05, 65W05

1. Introduction. We discuss parallel algorithms for solving linear systems of the form

$$Ax = b, \quad A \in R^{n \times n}, \quad \text{nonsingular}, \quad A_{ij} = 0 \quad \text{if } |i-j| > k, \tag{1.1}$$

where k is an integer with $k < n$ (usually $k \ll n$). In the usual terminology, k is said to be the half-bandwidth of A; the bandwidth is $2k+1$. Such systems arise in many numerical algorithms, for example, in the numerical solution of partial differential equations and of optimal control problems. Often the system has other characteristics which can be exploited, e.g., it is actually block-tridiagonal, or is known to be symmetric positive definite, in which case more specialized algorithms than the one to be described here may be appropriate.

Much effort has been devoted to solving special cases of (1.1) on parallel machines; see, for example, Johnsson [10]; van der Vorst [15]; Buzbee, Golub, and Nielson [3]; Hockney and Jesshope [9]; Sameh and Kuck [14]; and Berry and Sameh [1] for tridiagonal and block-tridiagonal systems, and Lawrie and Sameh [12] and Dongarra and Johnsson [5] for banded systems. Most of these papers describe algorithms which can fail to produce valid solutions even when the problem is well conditioned and exact arithmetic is used. Two exceptions are the algorithm in [14], and the "double-width separator" algorithm described by Gilbert and Schreiber [6] and Conroy [4], which will be discussed later. Most of the algorithms can be expressed in terms of a "tearing" of the initial matrix, and subsequent application of the Sherman-Morrison-Woodbury formula for inverse updating. In the case of the well-known cyclic reduction method (see Buzbee, Golub, and Nielson [3]), this tearing is applied recursively, and the size of the remaining subsystem is halved at each level. The problem with many of these methods arises from the fact that the tearing strategy corresponds to choosing a predetermined elimination ordering for the variables, which is based on considerations of efficient implementation rather than numerical validity. For many special cases of (1.1) (e.g., those in which A is symmetric positive definite) this presents no difficulty. However, it is our aim here to specify an algorithm which will, at least in principle, produce a valid solution for *all* problems of the form (1.1).

* Received by the editors March 8, 1989; accepted for publication (in revised form) May 9, 1990. This research was supported in part by National Science Foundation grants ASC-8714009, DMS-8619903, and DMS-8900984, and Air Force Office of Scientific Research grant AFOSR-ISSA-87-0092.

† Mathematics Department, Box 8205, North Carolina State University, Raleigh, North Carolina 27695-8205. This research was performed while the author was visiting the Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439.

Our algorithm is based on that of Dongarra and Johnsson [5], in which the matrix is broken up along the diagonal into a number (say, p) of blocks, each separated by a small dense $k \times k$ block. Each main block has the same bandwidth as the original matrix A, and each is factorized using Gaussian elimination without pivoting. A reduced system of dimension approximately $(p-1)k$ is then formed from the rows of A corresponding to the separating blocks. This is solved, and the remaining variables are recovered using data stored during the initial factorization stage.

The main difference in our method is to allow pivoting to occur in the factorization of each of the main blocks. Row pivoting alone is not enough, as these blocks can be rank deficient—they can have nullity as high as $2k$. Considerable complication is therefore introduced in the formation of the reduced system. When singularity is detected in any of the blocks, the dimension and bandwidth of the reduced system is increased. After solution of the reduced system, the remaining variables are recovered as before. Despite its greater complexity, the potential for speedup of the algorithm, when implemented on a multiprocessor, remains high. Most of the computation takes place in the initial factorization of the main diagonal blocks, which is "embarrassingly parallel."

The remainder of the paper is organized as follows. In § 2, the algorithm is described under assumptions of exact arithmetic, and its validity is proved. In § 3, we specify a practical version, perform an error analysis, and estimate the computational requirements. The double-width separator method is discussed briefly in § 4. Implementation of both algorithms on shared-memory multiprocessors at the Argonne Advanced Computing Research Facility is described in § 5.

Throughout the paper, superscripts on vector and matrix quantities are used to denote partition number, while subscripts are used for components. On scalar quantities, a subscript is used for partition number. The notation $\|\cdot\|$ denotes $\|\cdot\|_\infty$ unless otherwise specified.

2. Fundamentals.

2.1. An outline. We start by imposing the same partitioning on the system as that used in Dongarra and Johnsson [5, Fig. 7]. The matrix is broken up diagonally into p main blocks, which are separated along the diagonal by smaller $k \times k$ separator blocks (see Fig. 1). We make the implicit assumption throughout that n is substantially greater than $p^2 k$. This ensures that the size of the reduced system is significantly smaller than the size of the main diagonal blocks which are factorized concurrently, and so efficient multiprocessor implementation of the algorithm will be possible. For convenience, we assume that all main blocks A^ℓ have the same dimension m. The dimension of the entire system is therefore $n = pm + (p-1)k$.

Before describing the factorization of each A^ℓ, we note that these submatrices may be singular according to the following lemma.

LEMMA 2.1. *Under the assumptions in* (1.1), *each* A^ℓ, $\ell = 2, \cdots, p-1$ (*the "internal" partitions*) *has rank at least* $m - 2k$. *Any collection of* t_ℓ *columns taken from columns* $1, \cdots, m-k$ *of* A^ℓ *has rank at least* $t_\ell - k$. *Similarly, any collection of* t_ℓ *rows taken from rows* $1, \cdots, m-k$ *of* A^ℓ *has rank at least* $t_\ell - k$.

Proof. By extracting columns $(\ell-1)m + (\ell-1)k + 1, \cdots, \ell m + (\ell-1)k$ from the matrix A, and stripping off the zero rows, we obtain

$$\begin{bmatrix} T^{\ell 1} & 0 & \cdots & 0 \\ & & A^\ell & \\ 0 & 0 & \cdots & T^{\ell 2} \end{bmatrix}, \tag{2.1}$$

which has dimensions $(m \times 2k) \times m$ and rank m, according to (1.1). When we remove

the first k and last k rows, we are left with A^ℓ, which must therefore have rank at least $m-2k$.

By choosing a selection of t_ℓ columns from columns $1, \cdots, m-k$ of the matrix (2.1), we have

$$\begin{bmatrix} \tilde{T}^{\ell 1} \\ \tilde{A}^\ell \\ 0 \end{bmatrix}, \tag{2.2}$$

where $\tilde{T}^{\ell 1} \in R^{k \times t_\ell}$ consists of columns of $T^{\ell 1}$ and zero columns, while $\tilde{A}^\ell$ consists of t_ℓ columns from A. Again from (1.1), the matrix (2.2) has full column rank t_ℓ, and so by removing $\tilde{T}^{\ell 1}$ we conclude that $\tilde{A}^\ell$ has rank at least $t_\ell - k$. An identical argument is used to prove the last statement of the lemma. □

Similar results apply to the "boundary" partitions A^1 and A^p, which both have rank at least $m-k$. For simplicity we restrict most of the following discussion to the internal partitions.

The factorization of each A^ℓ uses an enhanced row pivoting strategy. Lemma 2.1 implies that there may be stages of the factorization at which no suitable pivot is to be found in the current column. In this case it is necessary to look ahead to subsequent columns. As we show later, this look-ahead, denoted in the algorithm by *look*(i), is at most k columns, except possibly in the final stages of the factorization. We also show that, in the absence of rounding error, the values chosen for *look*(i) below are sufficient to ensure a valid factorization of A^ℓ under the assumptions (1.1). If it is found at stage i of the elimination that columns $i, i+1, \cdots, j-1$ (for some $j > i$) contain no suitable pivots, then a cyclic column permutation is performed: these columns are moved to the end of the matrix, and columns $j, j+1, \cdots, m$ are shifted $j-i$ places to the left.

The factorization algorithm makes use of a logical test *reject*$_i$, whose purpose is to test whether a prospective pivot element is too small to be used. In this section we define it very simply as follows:

$$\textit{reject}_i(\alpha) = \text{true} \Leftrightarrow \alpha = 0.$$

In the next section a more practical definition is supplied.

In the following statement of the algorithm, we use the integer $d(i)$ to denote the number of zero eigenvalues detected *before* stage i of the factorization. The notation *largest*$(\alpha_1, \alpha_2, \cdots)$ denotes the element of largest magnitude in the given argument list $\alpha_1, \alpha_2, \cdots$.

ALGORITHM **factor** (A^ℓ).
Initialize: $d(1) \leftarrow 0$.
for $i = 1, \cdots, m$
  if $i \leqq m-2k$
  then *look*$(i) \leftarrow k - d(i)$
  else *look*$(i) \leftarrow m - d(i) - i$;
  if *look*$(i) < 0$
  then STOP (A^ℓ has rank $i-1$)
  for $j = i, i+1, \cdots, i+$*look*(i)
    (look for pivot element in column j)
    $J \leftarrow \min(m, j+k+d(i))$;
    $A^\ell_{gj} \leftarrow$ *largest*$(A^\ell_{ij}, A^\ell_{i+1,j}, \cdots, A^\ell_{Jj})$;
    if *reject*$_i(A^\ell_{gj})$ **then** go to next j;
    if $j \neq i$
    then reorder the columns $(i, i+1, \cdots, m)$ of A^ℓ

```
        to (j, ···, m, i, i+1, ···, j−1);
        d(i+1) ← d(i)+(j−i);
    if g ≠ i
    then swap rows g and i;
    (now perform the elimination)
    for i_1 = i+1, ···, m
        m^ℓ_{i_1,i} ← A^ℓ_{i_1,i}/A^ℓ_{ii}; A^ℓ_{i_1,i} ← 0;
        for i_2 = i+1, ···, m
            A^ℓ_{i,i_2} ← A^ℓ_{i_1 i_2} − m^ℓ_{i_1,i} A^ℓ_{i,i_2}
        end (for i_2)
    end (for i_1)
    go to next i;
end (for j)
(at this stage, all possible pivots have been rejected)
if i > m−2k
then STOP (A^ℓ has rank i−1)
end (for i).
```

Before proving that the above algorithm is valid under the assumptions (1.1), we note that the integer J is defined in such a way that the elements $A^{\ell}_{J+1,j}, \cdots, A^{\ell}_{mj}$ are all zero. This is because

- The original matrix A^{ℓ} has lower bandwidth k,
- At stage i, the columns $i+1, \cdots, i+look(i)$ have been shifted a total of $d(i)$ places to the left on previous iterations, thus increasing the lower bandwidth to $k+d(i)$,
- Row pivoting and elimination at earlier stages has not increased the lower bandwidth beyond this.

THEOREM 2.2. *Under the assumptions* (1.1), *and given the partitioning in Fig.* 1, *the algorithm* **factor** *terminates successfully in the absence of rounding error, and produces a factorization of the form*

$$P^{\ell}A^{\ell}\Pi^{\ell} = \begin{bmatrix} L^{\ell} & 0 \\ L^{\ell}_{(2)} & I \end{bmatrix} \begin{bmatrix} U^{\ell} & U^{\ell}_{(2)} \\ 0 & 0 \end{bmatrix}, \tag{2.3}$$

where P^{ℓ} and Π^{ℓ} are permutation matrices, L^{ℓ} and U^{ℓ} have dimensions $m_{\ell} \times m_{\ell}$, m_{ℓ} is the rank of A^{ℓ}, and the other submatrices in the partition are dimensioned accordingly. L^{ℓ} and U^{ℓ} are upper and lower triangular, respectively.

Proof. We first show that for all $i \leqq m-2k$, we have $look(i) \geqq 0$, and a pivot will be found somewhere in columns $i, i+1, \cdots, i+look(i)$ during the inner loop. This is done by contradiction. Suppose for the moment that $look(i) \geqq 0$, and that after $i-1$ stages, $A^{\ell}_{gj} = 0$, for all $g = i, \cdots, m$ and $j = i, \cdots, i+look(i)$, where $look(i) = k - d(i)$. Let $\hat{A}^{\ell}$ be the matrix consisting of the first $i+k$ columns of A^{ℓ}. By Lemma 2.1, $\hat{A}^{\ell}$ has rank at least i. Let L^{ℓ}_{i-1} be the $m \times m$ product of the first $i-1$ Gauss transformations which were applied at previous elimination steps. Since L^{ℓ}_{i-1} is nonsingular, the rank of $L^{\ell}_{i-1}\hat{A}^{\ell}$ will also be at least i. However, each column of the latter matrix has zeros in positions $i, \cdots, m$ because either

- It has been a pivot column at an earlier stage of the factorization,
- It had been pivoted to the end at an earlier stage, say stage $j < i$, because it had zeros in positions $j, \cdots, m$. This property will not have changed on subsequent steps, or
- It falls within the range of the current look-ahead. Columns $i, \cdots, i+k-d(i)$ at stage i were originally in positions $i+d(i), \cdots, i+k$ of the unreduced A^{ℓ}.

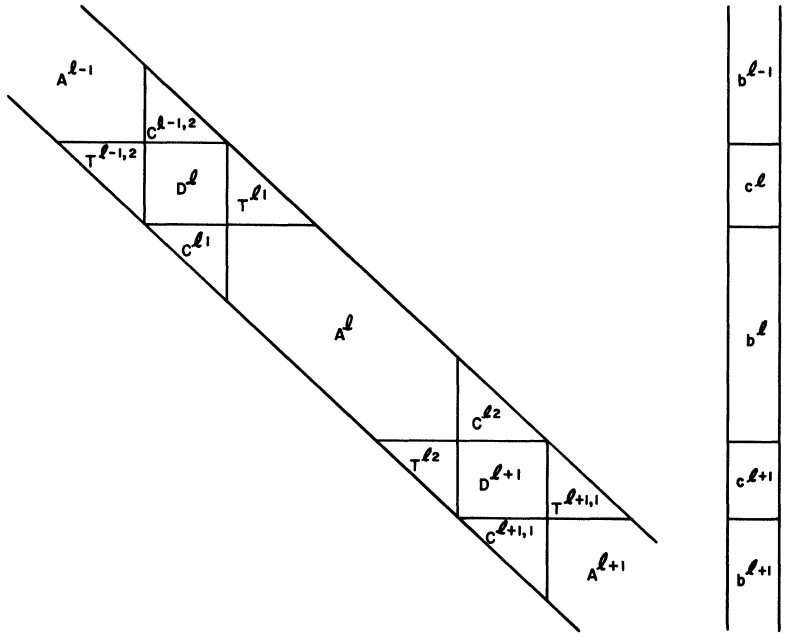

FIG. 1. *Initial partitioning of the matrix A and right-hand side b.*

Hence $L_{i-1}^{\ell}\hat{A}^{\ell}$ has only $i-1$ rows which contain nonzeros, and so has rank at most $i-1$, giving a contradiction.

If $look(i)<0$ for some $i \leqq m-2k$ (i.e., $d(i)>k$ at stage i), we proceed exactly as above in showing that $L_{i-1}^{\ell}\hat{A}^{\ell}$ has only zeros in rows $i, \cdots, m$. The only difference is that the third case no longer applies. A contradiction is obtained again.

It follows from this that **factor** does not terminate until $i > m-2k$. It only remains to show that the triangular factors have the form indicated in (2.3). From the usual properties of Gaussian elimination, it is clear that the factors are indeed triangular. We only need show that the lower right blocks of both factors have the desired structure.

When $m_\ell = m$, no column pivoting is necessary, and the algorithm reduces to standard Gaussian elimination with row partial pivoting. In this case, the conclusion of the theorem is clearly true.

When $m_\ell < m$, one of the two premature exits may be taken. Suppose an exit occurs at stage i. From the first part of the proof, $i > m-2k$ and $look(i) = m-d(i)-i$. Because columns $m-d(i)+1, \cdots, m$ of the remaining matrix (i.e., the U factor) are columns which have been moved to the end at previous stages, they clearly contain zeros in row positions $i, \cdots, m$. If $look(i) \geqq 0$, it is also true (from the failure of the j-loop in **factor**) that columns $i, \cdots, m-d(i)$ also contain only zeros in row positions $i, \cdots, m$. In any case, there is a square zero block of dimension $(m-i+1)$ in the lower right corner of the U factor which, since $m_\ell = i-1$, is the desired form (2.3). For the L factor, the lower right block is the identity since no Gauss transformations are applied during stages $i, \cdots, m$. □

When A^{ℓ} has full rank, it is well known that when Gaussian elimination with row partial pivoting is applied, the factor L^{ℓ} can be stored in a data structure with lower bandwidth k, and U^{ℓ} will have upper bandwidth $2k$. The analogous result when our pivot strategy is used is as follows.

THEOREM 2.3. *Under the assumptions in* (1.1), *the Gauss transformations which make up* L^ℓ *and* $L^\ell_{(2)}$ *in* (2.3) *each contain at most* $2k$ *subdiagonal elements, and* U^ℓ *has upper bandwidth* $2k$.

Proof. From (2.3), we see that L^ℓ, $L^\ell_{(2)}$, and U^ℓ can be considered as arising from applying Gaussian elimination with row partial pivoting to the $m \times m_\ell$ matrix

$$A^\ell \bar{\Pi}^\ell, \tag{2.4}$$

where $\bar{\Pi}^\ell$ consists of the first m_ℓ columns of Π^ℓ. The result for the lower triangular factor will follow if we can show that the lower bandwidth of the matrix (2.4) is at most $2k$. It is clearly sufficient to establish this for the first $m-2k$ columns of the matrix. To see this, observe that columns $1, \cdots, m-2k$ of (2.4) consist of columns of A^ℓ which have been shifted at most $d(m-2k+1)$ places to the left (where d is as defined in **factor**). Since $d(m-2k+1) \leqq k$, the lower bandwidth of these columns cannot have increased to more than $2k$, giving the result for L^ℓ.

We prove the result for U^ℓ inductively by rows, again with reference to (2.4). More specifically, it is shown that after each step i of the elimination, the remaining $(m-i) \times (m_\ell - i)$ submatrix satisfies the following property:

> "For each row of the submatrix, if a nonzero appears in column position j, then all columns after column position $j+2k$ will contain zeros."

This property simply means that there are at most $2k+1$ nonzeros in each row, and that they occur within a band. If the property holds for each of the m_ℓ submatrices which arise during the elimination process, we have the desired result. This is because, possibly after a row swap, the pivot position (i.e., the top left position of the submatrix) contains a nonzero, and so the pivot row has upper bandwidth at most $2k$.

Clearly the property is satisfied by the initial matrix (2.4). Assuming it is true for the submatrix which remains at stage i, we aim to show that it is true for the next step.

After finding a pivot at stage i by looking down the first column of the submatrix, and doing a row swap if necessary, we have already observed that the pivot row has upper bandwidth at most $2k$. For each nonzero in column 1 of the submatrix, we eliminate by adding a multiple of the pivot row. This does not disturb the property. When we form the new submatrix for stage $i+1$ by deleting the first row and column, it is easy to see that the property still holds. Hence the upper bandwidth of U^ℓ is at most $2k$. □

The algorithm **factor** is of course only a part of the overall method for solving (1.1). We now show how it is embedded into the remainder of the outer algorithm. Each major computational step is referred to as a "phase."

Phase 1. Apply **factor** to A^ℓ, $\ell = 1, \cdots, p$, and simultaneously apply the row pivots and forward-elimination steps to the right-hand side b^ℓ and the left and right flanking matrices $C^{\ell 1}$ and $C^{\ell 2}$. In addition, apply the column permutations to the top and bottom flanking matrices $T^{\ell 1}$ and $T^{\ell 2}$, and after each step i of **factor**, eliminate the elements in column i of the top and bottom flanks. This step is explained in more detail below. Figure 2 gives an indication of the possible appearance of a partition at the end of this phase.

Phase 2. Form a reduced system by taking the "separating" rows and columns and the "degenerate" rows and columns (positions $m_\ell+1, \cdots, m$) from each partition. The resulting system has the block-pentadiagonal form shown in Fig. 3. Its dimension is

$$(p-1)k + \sum_{\ell=1}^{p} (m - m_\ell),$$

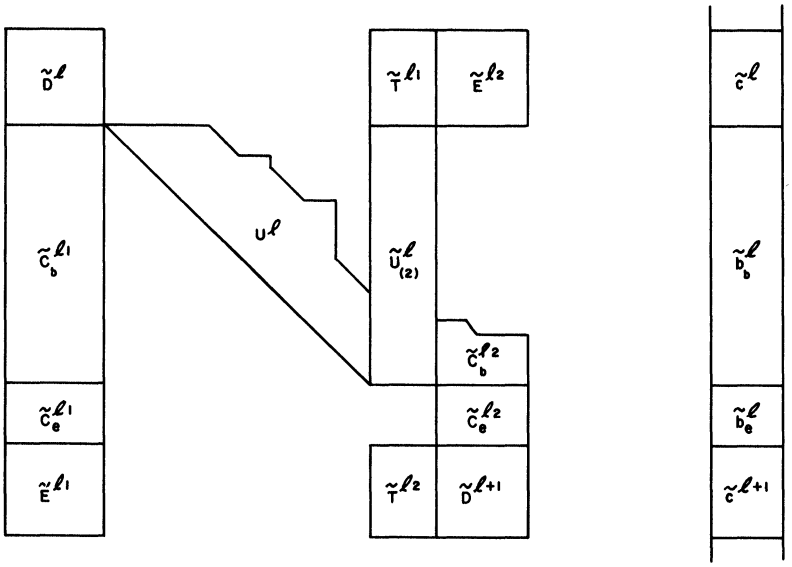

FIG. 2. *Partitioning ℓ after Phase 1 of the algorithm.*

$$\begin{bmatrix} \ddots & \ddots & \ddots & & & & & \\ \tilde{E}^{\ell-1,1} & \tilde{T}^{\ell-1,2} & \tilde{D}^{\ell} & \tilde{T}^{\ell 1} & \tilde{E}^{\ell 2} & & & \\ & & \tilde{C}_e^{\ell 1} & \tilde{S}^{\ell} & \tilde{C}_e^{\ell 2} & & & \\ & & \tilde{E}^{\ell 1} & \tilde{T}^{\ell 2} & \tilde{D}^{\ell+1} & \tilde{T}^{\ell+1,1} & \tilde{E}^{\ell+1,2} & \\ & & & & \tilde{C}_e^{\ell+1,1} & \tilde{S}^{\ell+1} & \tilde{C}_e^{\ell+1,2} & \\ & & & & & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ y_c^{\ell} \\ y_e^{\ell} \\ y_c^{\ell+1} \\ y_e^{\ell+1} \\ y_c^{\ell+2} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \tilde{c}^{\ell} \\ \tilde{b}_e^{\ell} \\ \tilde{c}^{\ell+1} \\ \tilde{b}_e^{\ell+1} \\ \vdots \end{bmatrix}$$

FIG. 3. *Structure of the reduced system.*

while its bandwidth is

$$2k-1+\max_{\ell=1,\cdots,p}(m-m_{\ell}).$$

(In the absence of rank-deficiency in the blocks A^{ℓ}, the $\tilde{C}_e$, $\tilde{S}$, and $\tilde{T}$ blocks in Fig. 3 disappear, leaving a block-tridiagonal reduced system.) Solve the reduced system to obtain the vector

$$(\cdots,(y_c^{\ell})^T,(y_e^{\ell})^T,(y_c^{\ell+1})^T,\cdots)^T,$$

where $y_c^{\ell}\in R^k$ and $y_e^{\ell}\in R^{m-m_{\ell}}$.

Phase 3. Back-substitute to recover the eliminated variables:

$$y_b^{\ell}=(U^{\ell})^{-1}(\tilde{b}_b^{\ell}-\tilde{C}_b^{\ell 1}y_c^{\ell}-\tilde{C}_b^{\ell 2}y_c^{\ell+1}-\tilde{U}_{(2)}^{\ell}y_e^{\ell}). \tag{2.5}$$

Perform column permutations to recover the solution x:

$$x_c^{\ell}=y_c^{\ell}, \qquad x^{\ell}=\Pi^{\ell}\begin{bmatrix} y_b^{\ell} \\ y_e^{\ell} \end{bmatrix}.$$

The obvious way to execute this algorithm in a parallel environment is to assign each partition A^{ℓ} to a processor. Each factorization in Phase 1 and each back-substitution in Phase 3 can then be carried out concurrently. Phase 2, however, is usually carried out on a single processor.

As described in Phase 1, elimination of the elements in the top and bottom flanks is interleaved with the Gaussian elimination process applied to the main blocks. However, we can conceptually view this process as happening *after* the algorithm **factor** has done its work, that is, after U^ℓ has been found, and the left and right flanks and right-hand side have been modified to produce the matrices $\tilde{C}^{\ell 1}$, $\tilde{C}^{\ell 2}$, and $\tilde{b}^\ell$, partitioned as shown in Fig. 2. Elimination of the top flank $T^{\ell 1}$ is achieved by first performing the column pivot sequence used by **factor**:

$$\bar{T}^{\ell 1} = [T^{\ell 1} \quad 0 \quad \cdots \quad 0]\Pi^\ell = [\bar{T}_b^{\ell 1} \quad \bar{T}_e^{\ell 1}]$$

(where $\bar{T}_b^{\ell 1}$ consists of the first m_ℓ columns), and then finding a $k \times m_\ell$ matrix $M_b^{\ell 1}$ such that

$$M_b^{\ell 1} U^\ell + \bar{T}_b^{\ell 1} = 0.$$

The other top-flank matrices in Fig. 2 are then produced by the formulae

$$\begin{aligned} \tilde{D}^\ell &\leftarrow D^\ell + M_b^{\ell 1} \tilde{C}_b^{\ell 1}, \\ \tilde{T}^{\ell 1} &= \bar{T}_e^{\ell 1} + M_b^{\ell 1} U_{(2)}^\ell, \\ \tilde{E}^{\ell 2} &= M_b^{\ell 1} \tilde{C}_b^{\ell 2}, \\ \tilde{c}^\ell &\leftarrow c^\ell + M_b^{\ell 1} \tilde{b}_b^\ell. \end{aligned} \tag{2.6}$$

For the bottom flank, we find another matrix of multipliers $M_b^{\ell 2}$ and obtain $\tilde{E}^{\ell 1}$, $\tilde{T}^{\ell 2}$, $\tilde{D}^{\ell+1}$, and $\tilde{c}^{\ell+1}$ in the same way. Note the $\tilde{D}^\ell$, $\tilde{D}^{\ell+1}$, $\tilde{c}^\ell$, and $\tilde{c}^{\ell+1}$ are also updated by the partitions $\ell - 1$ and $\ell + 1$ during Phase 1.

Phase 1 may also be described in terms of an initial red-black reordering of the matrix, which takes the initial form

$$A = \begin{bmatrix} A^1 & C^{12} & & & & & & & \\ T^{12} & D^1 & T^{21} & & & & & & \\ & C^{21} & A^2 & C^{22} & & & & & \\ & & T^{22} & D^2 & T^{31} & & & & \\ & & & C^{31} & A^3 & C^{32} & & & \\ & & & & \ddots & \ddots & \ddots & & \\ & & & & & & T^{p-1,2} & D^{p-1} & T^{p,1} \\ & & & & & & & C^{p1} & A^p \end{bmatrix}$$

to the reordered form

$$P_0 A \Pi_0 = \left[\begin{array}{ccccc|ccccc} A^1 & & & & & C^{12} & & & & \\ & A^2 & & & & C^{21} & C^{22} & & & \\ & & \ddots & & & & \ddots & \ddots & & \\ & & & A^{p-1} & & & & C^{p-1,1} & C^{p-1,2} & \\ & & & & A^p & & & & C^{p1} & \\ \hline T^{12} & T^{21} & & & & D^1 & & & & \\ & T^{22} & T^{31} & & & & D^2 & & & \\ & & \ddots & \ddots & & & & \ddots & & \\ & & & T^{p-1,2} & T^{p1} & & & & D^{p-1} & \end{array}\right].$$

During Phase 1, additional row and column pivoting is needed during the factorization of each A^ℓ. Also, the blocks created by the "rejected" columns need to be

pivoted out of the first part. Using the notation introduced above, the net result is a factorization

$$P_1 A \Pi = \bar{L}\bar{U},$$

where

$$\bar{L} = \left[\begin{array}{cccc|c} L^1 & & & & \\ & L^2 & & & 0 \\ & & \ddots & & \\ & & & L^p & \\ \hline L^1_{(2)} & & & & I \\ M_b^{12} & M_b^{21} & & & \\ & \ddots & & & \ddots \\ & & M_b^{p-1,2} & M_b^{p1} & \\ & & & L^p_{(2)} & I \end{array}\right],$$

$$\bar{U} = \left[\begin{array}{cccc|cccc} U^1 & & & & U^1_{(2)} & \tilde{C}_b^{12} & & \\ & U^2 & & & & \tilde{C}_b^{21} & & \\ & & \ddots & & & & \ddots & \\ & & & & & & \tilde{C}_b^{p-1,2} & \\ & & & U^p & & & \tilde{C}_b^{p1} & U^p_{(2)} \\ \hline & 0 & & & & \hat{A} & & \end{array}\right].$$

The Schur complement $\hat{A}$, which is the coefficient matrix of the reduced system (Fig. 3), is then factorized using row partial pivoting:

$$P_2 \hat{A} = \hat{L}\hat{U},$$

yielding an overall factorization

$$PA\Pi = \tilde{L}\tilde{U}, \tag{2.7}$$

where

$$P = \begin{bmatrix} I & 0 \\ 0 & P_2 \end{bmatrix} P_1.$$

3. Practical implementation.

3.1. Pivot suitability. Clearly, the test *reject*$_i$ for suitability of a candidate pivot element is insufficient if we want the algorithm to retain numerical stability properties. For instance, during the elimination of a column-i element in a top or bottom flank, a large multiple will be needed if this element is much larger than the pivot being used to eliminate it. This means that the norm of the lower triangular factor in the LU decomposition which we are implicitly forming in Phase 1 will be too large. Formally, let $\bar{T}^{\ell 1,i}$ and $\bar{T}^{\ell 2,i}$ be the $k \times m$ top and bottom flanking matrices, after $i-1$ elimination steps. If a_{ii}^{ℓ} is the prospective pivot for the ith stage of the Phase 1 elimination, *reject*$_i(a_{ii}^{\ell})$ is set to *false* only if the following inequalities hold:

$$\begin{aligned} \left|\frac{\bar{T}_{ji}^{\ell 1,i}}{a_{ii}^{\ell}}\right| &\leqq \mu, \qquad j = 1, \cdots, k, \\ \left|\frac{\bar{T}_{ji}^{\ell 2,i}}{a_{ii}^{\ell}}\right| &\leqq \mu, \qquad j = 1, \cdots, k, \end{aligned} \tag{3.1}$$

where $\mu > 1$ is a chosen constant. If (3.1) fails, then column i is swapped to the end of the matrix as described in **factor**, causing the dimension of the reduced system to increase by one. Because a_{ii}^{ℓ} and the elements below it may not be exactly zero, the block A^{ℓ} is not necessarily rank-deficient, since the lower right block of the U factor in (2.3) need not be zero. However, **factor** will still have the behavior discussed in § 2 provided μ is not chosen so small that more than k columns are rejected during the first $m-2k$ stages of the factorization. As expected, we find in the error analysis that a smaller μ leads to better numerical stability, while the increased rejection rate causes loss of efficiency (see § 5).

3.2. Error analysis. Since the algorithm can be thought of as Gaussian elimination with a certain restricted pivot strategy, we can use the standard backward error analysis framework. The following theorem can be immediately deduced from, for example, Theorem 4.3-2 of Golub and Van Loan [7].

THEOREM 3.1. *Let $\tilde{L}\tilde{U}$ be the factorization of $PA\Pi$ computed by Gaussian elimination, where P and Π are arbitrary permutation matrices. Then the solution $\hat{x}$ of the system $Ax = b$ which is computed using this factorization is the exact solution of a perturbed system $(A+E)\hat{x} = b$, where*

$$\|E\|_\infty \leqq \phi_1(n)[\|A\|_\infty + \|\tilde{L}\|_\infty \|\tilde{U}\|_\infty]u, \tag{3.2}$$

where $\phi_1(n) = O(n)$ and u is unit roundoff.

Since our aim is to bound the second term in (3.2) in terms of $\|A\|_\infty$, we need to examine element growth during the factorization. Phases 1 and 2 are considered separately.

Referring to the notation in (2.7), a bound for $\|\tilde{L}\|_\infty$ can be deduced by noting that the elements of the L^{ℓ}, $L_{(2)}^{\ell}$, and $\hat{L}$ blocks are all bounded in magnitude by one, while from (3.1), those in $M_b^{\ell 1}$ and $M_b^{\ell 2}$ are bounded by μ. Since each $M_b^{\ell 2}$ has at most k nonzero columns, we deduce that

$$\|\hat{L}\|_\infty \leqq (m+k)\mu + m_r, \tag{3.3}$$

where m_r is the order of the reduced system.

Growth in the $\bar{U}$ factor during the first phase can, unfortunately, be exponential when $k > 1$. To be more specific, we need to define the order-k Fibonacci sequence: For $k \geqq 1$, define

$$F(i,k) = 2^{i-1} \quad \text{for } i = 1, \cdots, k,$$

$$F(i,k) = \sum_{j=1}^{k} F(i-j,k) \quad \text{for } i > k.$$

We also observe that for any $m \geqq k > 1$,

$$\sum_{i=1}^{m} F(i,k) \leqq \left\lceil \frac{\log km}{\log k} \right\rceil F(m,k). \tag{3.4}$$

(Here, $\lceil z \rceil$ represents the smallest integer which is larger than z.) This can be proved by first extending the definition above to set $F(i,k) = 0$ for $i \leqq 0$. Then for any integer i,

$$\sum_{j=1}^{k} F(i-j,k) \leqq F(i,k).$$

Choosing t so that $k^{t-1} < m \leqq k^t$, we find, by successively grouping together sets of k consecutive terms, that

$$\begin{aligned}\sum_{i=1}^{m} F(i,k) &= \sum_{j=0}^{k^t} F(m-j,k) \\ &= F(m,k) + \sum_{j=0}^{k^{t-1}} F(m-jk,k) \\ &\leqq 2F(m,k) + \sum_{j=0}^{k^{t-2}} F(m-jk^2,k) \\ &\vdots \\ &\leqq (t+1)F(m,k).\end{aligned}$$

In the tridiagonal case ($k=1$), we have $F(i,1)=1$ for $i>0$, and hence $\sum_{i=1}^{m} F(i,1) = m$.

First we consider the case in which no column pivoting is needed during the factorizations in Phase 1. The number $F(m,k)$ bounds the element growth in the matrices $\tilde{C}_b^{\ell 1}$ and $\tilde{C}_b^{\ell 2}$, during Phase 1. This bound is sharp—the worst case occurs when initially $C_{i,k}^{\ell 1} = 1$ for $i = 1, \cdots, k$, and all multipliers on subsequent elimination steps are one. Element growth can also occur in the blocks which make up the reduced system, because of the elimination of top and bottom flanks (see (2.6)). At each elimination step, a multiple of up to μ of row i of $\tilde{C}_b^{\ell 1}$ and $\tilde{C}_b^{\ell 2}$ is added to D^{ℓ} and $D^{\ell+1}$, respectively. This yields an upper bound on growth in these blocks of

$$1 + 2\mu \sum_{i=1}^{m} F(i,k) \leqq 1 + 2\mu \left\lceil \frac{\log km}{\log k} \right\rceil F(m,k). \tag{3.5}$$

Growth in the blocks $\tilde{E}^{\ell 1}$ and $\tilde{E}^{\ell 2}$ is also bounded by this quantity. Growth in the upper triangular factors U^{ℓ} themselves is not as significant. Bohte [2, p. 140] shows that when Gaussian elimination with partial pivoting is applied to a matrix with half-bandwidth k, the growth factor is bounded by

$$R_k \overset{\text{def}}{=} 2^{2k-1} - (k-1)2^{k-2}. \tag{3.6}$$

Since we assume $m \gg k$, this number is clearly bounded by $F(m,k)$.

In the no-column-pivot case, the matrix for the reduced system is block-tridiagonal with $k \times k$ blocks. Since the bandwidth is $2k-1$, the result (3.6) could be used to bound additional element growth during Phase 2. A slightly better bound can be deduced by noting that when row partial pivoting is applied to this matrix, no nonzero can be swapped to a position more than $3k-1$ places above the main diagonal (i.e., $\hat{U}$ has upper bandwidth at most $3k-1$). Since the maximum element size at most doubles whenever elements are updated, and since each element is updated at most $3k-1$ times, the growth during Phase 2 is bounded by 2^{3k-1}. The results can be summarized as follows.

THEOREM 3.2. *If $k>1$, and no column pivoting is performed on any block during Phase 1 of the factorization, then the growth factor g defined by*

$$g \overset{\text{def}}{=} \frac{\max_{i,j=1,\cdots,n} \tilde{U}_{ij}}{\max_{i,j=1,\cdots,n} A_{ij}}$$

is bounded by

$$2^{3k-1}\left[1 + 2\mu \left\lceil \frac{\log km}{\log k} \right\rceil F(m,k)\right]. \tag{3.7}$$

The computed solution $\tilde{x}$ *then satisfies* $(A+E)\tilde{x}=b$, *where*

$$\|E\|_\infty \leqq \phi_2(n, p, k)\mu^2 2^{3k}\left\lceil \frac{\log km}{\log k}\right\rceil F(m, k)\|A\|_\infty u,$$

and $\phi_2(n, p, k)=O(kn^2/p)$.

Proof. Since each row of $\tilde{U}$ has at most $4k$ nonzeros, the ratio $\|\tilde{U}\|_\infty/\|A\|_\infty$ is bounded by $4kg$. The result is then obtained by combining (3.2), (3.3), and (3.7), and using the relation $m \approx n/p$. □

The result can be adapted to the tridiagonal case, by noting the following facts. From (3.6), growth of at most two can occur during factorization of a tridiagonal matrix with row partial pivoting; no growth occurs in left and right flanks, while growth in the elements which make up the reduced system is at most $1+2m\mu$. The bounds

$$\|\tilde{L}\|_\infty \leqq 2(p-1)+(m+1)\mu,$$
$$\|\tilde{U}\|_\infty \leqq 6(1+2m\mu)\|A\|_\infty$$

apply, and so we can state the following result.

THEOREM 3.3. *If a tridiagonal matrix is factorized and no column pivoting is necessary in any block, the computed solution* $\tilde{x}$ *is the exact solution of the perturbed system* $(A+E)\tilde{x}=b$, *where*

$$\|E\|_\infty \leqq \phi_3(n, p)\mu^2\|A\|_\infty u,$$

where $\phi_3(n, p)=O(n^3/p^2)$.

In the case in which column pivoting occurs, the increase in lower bandwidth of the rearranged matrix makes it necessary to consider order-$2k$ Fibonacci bounds. Growth in the left and right flanks, and in the "rejected" columns of A^ℓ is bounded by $F(m, 2k)$. Since the U^ℓ block has upper bandwidth at most $2k$, it can be shown that growth in these matrices is bounded by the smaller quantity 2^{2k}. Phase 1 growth in the blocks which make up the reduced system is bounded by

$$1+2\mu\left\lceil \frac{\log 2km}{\log 2k}\right\rceil F(m, 2k).$$

In Phase 2, the block-banded reduced system has half-bandwidth at most $4k-1$, and if it is factorized using row partial pivoting, the upper triangular factor produced will have half-bandwidth at most $7k-1$. Growth is thus bounded by 2^{7k-1}. The net result can be summarized as follows.

THEOREM 3.4. *If column pivoting is performed in Phase* 1, *then the growth factor* g *is bounded by*

$$2^{7k-1}\left[1+2\mu\left\lceil \frac{\log 2km}{\log 2k}\right\rceil F(m, 2k)\right]. \tag{3.8}$$

The computed solution $\tilde{x}$ *then satisfies* $(A+E)\tilde{x}=b$, *where*

$$\|E\|_\infty \leqq \phi_2(n, p, k)\mu^2 2^{7k}\left\lceil \frac{\log 2km}{\log 2k}\right\rceil F(m, 2k)\|A\|_\infty u,$$

and $\phi_2(n, p, k)=O(kn^2/p)$.

A few observations on these results are in order. First, the presence of the term $F(m, k)$ makes the growth factor exponential in the size of the submatrices. This contrasts with the situation for the one-processor algorithm of choice, namely, Gaussian elimination with row partial pivoting, which Bohte [2] shows yields a growth factor which is exponential in the bandwidth (3.6). It also contrasts with the situation for

general $n \times n$ matrices, for which the growth term can be exponential in n in the worst case. As in that case, we expect that this term is overly pessimistic and that the average-case performance should be much better, an assertion that the numerical results appear to support.

Second, we note the presence of the term μ^2 in the bound on $\|E\|_\infty$. As stated earlier, a larger choice of μ will decrease the need for column pivoting while it does not appear to have too deleterious an effect on the error bound. In fact, the presence of this term is the main difference between the bounds obtained in Theorems 3.2 and 3.4 and the bounds obtained by Conroy [4] for the double-width separator algorithm. (The log term can be discounted, and the remaining point of difference is a term 2^{4k} in his analysis, which compares to a term 2^{3k} in our best case and 2^{7k} in our worst case. Since this is a pessimistic term, and since we are most interested in smaller values of k, it is not particularly significant.)

3.3. Operation counts. Operation counts for banded system solvers are complicated by the use of pivoting and the resulting fill-in. Their usefulness as predictors of run time is limited because loop overheads, conditional tests, and data movements can add significantly to the run time, particularly when k is small. There is an additional complication in our case because the boundary partitions take less time to process than the interior partitions. The total operation count is a function of the number of partitions p, which increases sharply as p increases from one to four (because the savings due to the boundary conditions are shrinking), but only displays a marginal increase for larger values of p.

Below, we count the operations in the parallelizable phases (Phases 1 and 3), and in the serial phase (Phase 2) under certain conditions, and make a comparison with the operation count for one-processor Gaussian elimination with partial pivoting. In the expressions below, each arithmetic operation $(+, -, \times, /)$ counts as one "flop."

The simplest case is the one in which no row or column pivoting is required. Here, approximately $(4k+2)mk$ operations are needed to factor each interior block and apply the Gauss transformations to the left and right flanks. An additional $2mk^2$ may be needed to eliminate the top flank. In Phase 2, the block tridiagonal solve takes about $(\frac{14}{3}k^3+6k^2)(p-1)$ operations when pivoting is not used. Recovery of the eliminated variables in Phase 3 can be accomplished in about $4mk$ operations. In summary:

Partitioned algorithm, no pivoting:
parallel: $(6k+6)mk$ operations per partition
serial: $(\frac{14}{3}k^3+6k^2)(p-1)$ operations.

A corresponding count for one-processor Gaussian elimination yields:

Serial algorithm, no pivoting:
serial: $(2k+5)nk$ operations.

This tends to suggest a factor-of-three increase in total operation count in cases in which not much pivoting is needed, for example, when the matrix is close to being diagonally dominant.

If we consider the case in which row pivoting takes place, but no column pivoting, we find that in the worst case (i.e., the U^ℓ factor filling out to an upper bandwidth of $2k$), the Phase-1 operation count jumps to approximately $(10k+2)mk$ operations, and Phase 3 may require up to about $6mk$ operations. If we also allow row pivoting in the solution of the reduced system, a total of about $(\frac{23}{3}k^3-5k^2)(p-1)$ operations may be

needed. Summarizing:

Partitioned algorithm, row pivoting only:
parallel: $(10k+8)mk$ operations per partition
serial: $(\frac{23}{3}k^3-5k^2)(p-1)$ operations.

Serial algorithm, row pivoting:
serial: $(4k+7)nk$ operations.

When column pivoting is also allowed, the operation count increases substantially, due to both the increased lower bandwidth of the remaining submatrix and the presence of the rejected columns in the last k positions of the submatrix, to which the Gauss transformations should also be applied. In addition, the reduced system increases in size and bandwidth, so in the worst case we obtain:

Partitioned algorithm, row and column pivoting:
parallel: $(20k+12)mk$ operations per partition
serial: $(87k^3-7k^2)(p-1)$.

4. The double-width separator algorithm. An algorithm which doubles the size of the separator in order to more completely decouple the main blocks was apparently first proposed by Gilbert and Schreiber [6] in the context of decomposition of two-dimensional domains for elliptic partial differential equations. Since then it has been further developed and implemented on a hypercube architecture by Oleson and Gilbert [13]. The coefficient matrix A in this case is a discretized differential operator, and hence has additional structure within the band, but their description applies equally well to the general case. An independent derivation and error analysis of the algorithm appears in Conroy [4].

The advantage of using double-width separators is that the rows in the top and bottom flanks can be used as pivot rows during the factorization of each block, without affecting the top and bottom flanks of adjoining partitions. This removes the need for column pivoting. However, the operation count may be increased appreciably, since it is now possible for the left and right flanks to have $2k$ nonzero columns. The reduced system is block tridiagonal, with blocks of size $2k$ and constant dimension $2(p-1)k$ (possibly twice as large as the reduced system produced by the single-width separator algorithm of §§ 2 and 3), though the off-diagonal blocks are only half full. In [13], a parallel algorithm for solving the reduced system is proposed, but it can be unstable for general matrices A.

To be more specific, we give operation counts for the double-width separator algorithm in the case in which no row pivoting is needed either in the factorization of the individual blocks or in the factorization of the reduced system, and in the worst case in which the row pivoting creates maximal fill-in. In the former case, the parallel count is the same as in the no-pivot case in § 3, but the serial count may be slightly larger:

Double-width separator algorithm, no pivoting:
parallel: $(6k+6)mk$ operations per partition
serial: $(\frac{46}{3}k^3-4k^2)(p-1)$ operations.

Double-width separator algorithm, row pivoting:
parallel: $(12k+10)mk$ operations per partition
serial: $(\frac{76}{3}k^3-6k^2)(p-1)$ operations.

These counts indicate that when no column pivoting is needed in the single-width separator, the serial part of the code requires about a third as many operations as for the double-width separator case, while execution time for the parallel parts are similar.

Therefore, if the reduced system is *not* small relative to the size of each partition, the double-width separator algorithm may be slower.

5. Numerical results. The single-width separator algorithm of §§ 2 and 3, and the double-width separator algorithm were implemented on two shared-memory machines at the Advanced Computing Research Facility, at Argonne National Laboratory. They were the Alliant FX/8 (8 processors) and the Sequent Balance (24 processors). On the Alliant, each processor has vector capabilities; these are not much help for our problem, except during the back substitution in Phase 3, which accounts for only a small part of the computation.

The cautionary remarks in § 3.3, regarding the usefulness of operation counts as predictors of runtime, are borne out by our numerical experience. The codes for the partitioned algorithms are not optimal in terms of their flop counts. To achieve this it is necessary to execute a number of comparisons and conditional tests, which usually use more cpu time than they save. Loop overheads are also significant, since for narrow-banded systems many loops are short and execute for only a few iterations.

For similar reasons, block algorithms and BLAS3 primitives are of limited applicability for narrow-banded systems. Any matrix-matrix operations would involve small blocks of dimension k or $2k$. Data locality in these algorithms is also good, as factorization of each block, and the reduced system, takes place in a reasonably sequential fashion. In the parallel implementation, the amount of working storage needed at any point in the execution of Phase 1 is a fairly small multiple of pk^2, and for the examples given here, the Alliant cache can easily meet these needs.

On both machines, Phases 1 and 3 were usually executed in parallel by assigning one block A^ℓ to each processor. Phase 2 (solution of the reduced system) was carried out on a single processor, using the LINPACK routines `dgbfa` and `dgbsl`. On the Alliant, these routines were compiled with the `-Ogv -DAS` options. Full optimization was found to be a hindrance rather than a help, as the results obtained by applying the LINPACK routines to solve the entire system (Table 8) show. Compiler directives were used extensively on the Alliant to "tune" both the single-width and double-width separator codes. Run time was improved by disabling vectorization on many short loops. The value $\mu = 10^4$ was used for the pivot tolerance in the single-width separator code.

The codes were tested on seven problems for which each partition A^ℓ in the single-width separator case was nonsingular, and so no column pivoting was necessary. The dimensions of these problems were as follows:

1. $n = 2{,}000$, $k = 2$,
2. $n = 2{,}000$, $k = 5$,
3. $n = 5{,}000$, $k = 2$,
4. $n = 5{,}000$, $k = 5$,
5. $n = 10{,}000$, $k = 2$,
6. $n = 10{,}000$, $k = 5$,
7. $n = 10{,}000$, $k = 8$.

The nonzero elements of the matrices were chosen from a uniform distribution in the interval $[-1, 1]$, except for the diagonals, which were chosen from the interval $[-.1, .1]$. The solution x was chosen to be the vector $(1, 2, 3, \cdots)^T$.

Results from the Balance on problems 1-4, using both the single-width and double-width separator codes, are given in Tables 1 and 2. In all cases, the number of processors p_n and number of partitions p were chosen to be identical. Unfortunately, core storage limits made it impossible to run the larger problems on this machine. The

TABLE 1
Balance results, single-width separator algorithm, nondegenerate banded problems. Times in seconds.

$p = p_n$	Prob. 1	Prob. 2	Prob. 3	Prob. 4
1	3.13	8.38	7.83	20.8
2	3.38	9.67	8.33	24.3
4	1.80	5.22	4.50	12.9
8	.98	2.88	2.35	6.75
16	.65	1.98	1.37	3.93
Max speedup over $p = 1$	4.8	4.2	5.7	5.3

TABLE 2
Balance results, double-width separator algorithm, nondegenerate banded problems. Times in seconds.

$p = p_n$	Prob. 1	Prob. 2	Prob. 3	Prob. 4
1	3.13	8.38	7.83	20.8
2	3.60	11.52	8.87	29.1
4	1.85	6.00	4.48	14.8
8	1.03	3.45	2.40	7.83
16	.70	2.80	1.40	4.93
Max speedup over $p = 1$	4.5	3.0	5.6	4.2

results follow the predicted behavior from the operation counts in §§ 3.3 and 4 quite closely, except that the multiprocessor counts are somewhat pessimistic for both codes. As noted in those sections, the total fill-in (and hence total operation count) increases sharply as p increases from 1 to 4, as the savings made in processing the first and last partitions diminish. In all cases the total cpu requirements more than double from $p = 1$ (standard Gaussian elimination with row partial pivoting) to $p = 2$, though this is ameliorated by spreading the work over two processors. For higher values of p, near-linear speedup is seen, until the serial bottleneck created by the reduced system results in a decrease in efficiency. For the pentadiagonal systems ($k = 2$) this is less of a problem than for the larger-bandwidth systems, which is not surprising as the size of the reduced system is about pk in the single-width case and $2pk$ in the double-width case. Comparing runtimes for the single- and double-width cases, we see that there are only minor differences for the $k = 2$ problems, while for the $k = 5$ problems the single-width code holds a distinct advantage which tends to grow as p increases. Table 4 gives the timings for the LINPACK routines on a single processor. These are similar to the 1-processor implementation reported in Tables 1 and 2.

Timings obtained by running problems 1, 2, 4, 5, 6, and 7 on the Alliant are reported in Tables 5 and 6. The first line of each table shows results for the serial algorithm (these can be compared with the LINPACK timings in Table 8). The second line shows results for an eight-partition version as run on a single processor, and the third line shows an eight-partition version on eight processors. Comparison of the second and third lines gives an indication of the multiprocessing efficiency; this is typically around 84 percent for the single-width code and between 70 and 81 percent for the double-width code. In each case, single-processor solution of the reduced system prevents higher efficiencies from being attained. Despite the fact that the same code was used for the Phase 1 and Phase 3 routines as on the Balance, the relative

TABLE 3
Balance results, single-width separator algorithm, degenerate banded problems. Times in seconds.

$p = p_n$	Prob. 1a	Prob. 2a	Prob. 4a
1	3.13	8.38	20.8
2	3.38	9.73	24.4
4	1.80	5.20	12.9
8	1.35	3.45	8.07
16	.83	2.18	4.40

TABLE 4
Balance results, LINPACK, 1 *processor, times in seconds.*

	Prob. 1	Prob. 2	Prob. 3	Prob. 4
LINPACK (1 processor)	3.47	8.35	8.67	20.7

TABLE 5
Alliant results, single-width separator algorithm, nondegenerate banded problems. Times in seconds.

p	p_n	Prob. 1	Prob. 2	Prob. 4	Prob. 5	Prob. 6	Prob. 7
1	1	.32	.78	1.93	1.61	3.89	7.14
8	1	.67	1.85	4.56	3.33	9.18	18.14
8	8	.10	.28	.68	.49	1.37	2.64
Max speedup over $p = 1$		3.2	2.7	2.8	3.2	2.8	2.7

TABLE 6
Alliant results, double-width separator algorithm, nondegenerate banded problems. Times in seconds.

p	p_n	Prob. 1	Prob. 2	Prob. 4	Prob. 5	Prob. 6	Prob. 7
1	1	.32	.78	1.93	1.61	3.89	7.14
8	1	.47	1.56	3.89	2.39	7.81	16.42
8	8	.08	.28	.63	.38	1.21	2.59
Max speedup over $p = 1$		4.0	2.7	3.1	4.2	3.2	2.8

TABLE 7
Alliant results, single-width separator algorithm, degenerate banded problems. Times in seconds.

p	p_n	Prob. 1a	Prob. 2a	Prob. 4a
1	1	.32	.77	1.93
8	1	.69	1.90	4.73
8	8	.14	.35	.83

TABLE 8
Alliant results, LINPACK, *times in seconds.*

	Prob. 1	Prob. 2	Prob. 4	Prob. 5	Prob. 6	Prob. 7
LINPACK (-Ogv-DAS)	.56	1.02	2.57	2.81	5.13	6.29
LINPACK (-O-DAS)	.71	1.52	3.83	3.56	7.56	7.98

timings are quite different. A preference is shown for the double-width separator algorithm except in the larger-bandwidth problems, that is, problems 2 and 7. This is probably due to nonnumerical computational overhead of the type discussed above; the double-width code is shorter and less complex than the single-width code. Apparently, arithmetic processing on the Alliant is sufficiently fast that this overhead is more significant than on the Balance.

Tables 3 and 7 show results for test problems in which certain elements of the matrices were chosen to create singular partitions in the single-width algorithm. The dimensions are

1a. $n = 2{,}000$, $k = 2$,
2a. $n = 2{,}000$, $k = 5$,
4a. $n = 5{,}000$, $k = 5$.

Again, using m to denote the dimension of each partition and m_ℓ to denote its rank, each of these problems had $m_\ell = m$, except that $m_2 = m - 2$ when $p = 8$ and $m_3 = m - 1$ when $p = 16$. The rank-deficient partitions take considerably longer to factor than the nonsingular ones, and hence a load-balancing problem is created. Table 3 shows a sharp increase in runtime for the cases $p = 8$ and $p = 16$ over the corresponding entries in Table 1. However, for the larger-bandwidth matrices, the runtimes are still competitive with, and in some cases somewhat better than, the corresponding results for the double-width algorithm in Table 2. On the Alliant (Table 7), it is seen that the eight-partition algorithm is only slightly slower than in the nondegenerate case when run on a single processor. Unfortunately, all of the increase in runtime takes place on a single processor, as can be seen in the timings for the eight-processor version.

In all cases, the relative error in the computed answer was very small (at least 10 digits of accuracy in double-precision), and was apparently independent of p.

6. Conclusions. We have discussed and analysed partitioned Gaussian elimination algorithms for solving banded linear systems on multiprocessors. The results of tests on shared-memory machines with small to moderate numbers of processing elements confirms their usefulness.

For larger numbers of processors, the method as described above is not suitable. To see this, we let the problem size and number of processors p_n increase as in the definition of "scaled speedup" (see Gustafson, Montry, and Benner [8]). When $p = p_n$ is increased, while m (the blocksize) and k (half-bandwidth) are held constant, the size of the reduced system is $O(pk)$, and the cost of solving it on a single processor eventually dominates the cost of factoring each $m \times m$ partition concurrently. (In practice, it is reasonable to expect problems to scale in this way. For example, a user might double the number of gridpoints in a discretization of an optimal control problem if twice as many processors are available. This does not affect the bandwidth but doubles the dimension of the system.) A strategy for handling this decrease in efficiency would be to apply the method *recursively*, that is, to solve the *reduced* system in the same way as the original system, but on a smaller number of processors. (This concept is similar to cyclic reduction.) We do not explore the proposal further here, but simply

note that its difficulties include the possible compounding of the element growth discussed in § 3.2, and the fact that the reduced system has larger bandwidth than the original system.

Acknowledgments. I thank Andrea Hudson for her help with the Sequent Balance implementation, and a referee for some helpful comments.

REFERENCES

[1] M. BERRY AND A. H. SAMEH, *Multiprocessor schemes for solving block-tridiagonal linear systems*, Internat. J. Supercomput. Appl., 2 (1988), pp. 37–57.

[2] Z. BOHTE, *Bounds for rounding errors in the Gaussian elimination for band systems*, J. Inst. Math. Appl., 16 (1975), pp. 133–142.

[3] B. L. BUZBEE, G. H. GOLUB, AND C. W. NIELSON, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal., 7 (1970), pp. 627–656.

[4] J. M. CONROY, *Parallel algorithms for the solution of narrow banded systems*, Appl. Numer. Math., 5 (1989), pp. 409–421.

[5] J. J. DONGARRA AND S. L. JOHNSSON, *Solving banded systems on a parallel processor*, Parallel Comput., 5 (1987), pp. 219–246.

[6] J. R. GILBERT AND R. SCHREIBER, *Nested dissection with partial pivoting*, in Proc. Sparse Matrix Symposium, Fairfield Glade, TN, 1982.

[7] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[8] J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BENNER, *Development of parallel methods for a 1024-processor hypercube*, SIAM J. Sci. Statist. Comput., 9 (1984), pp. 609–638.

[9] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, Bristol, U.K., 1981.

[10] S. L. JOHNSSON, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[11] H. B. KELLER, *Accurate difference methods for nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 11 (1974), pp. 305–320.

[12] D. H. LAWRIE AND A. H. SAMEH, *The computation and communication complexity of a parallel banded system solver*, ACM Trans. Math. Software, 10 (1984), pp. 185–195.

[13] T.-H. OLESON AND J. R. GILBERT, *One-way dissection with pivoting on the hypercube*, submitted for publication, 1989.

[14] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25 (1978), pp. 81–91.

[15] H. A. VAN DER VORST, *Large tridiagonal and block tridiagonal linear systems on vector and parallel computers*, Parallel Comput., 5 (1987), pp. 45–54.