The stable parallel solution of general narrow banded linear systems

THE STABLE PARALLEL SOLUTION OF GENERAL NARROW BANDED LINEAR SYSTEMS

PETER ARBENZ AND MARKUS HEGLAND

ABSTRACT. We propose a stable algorithm for the parallel solution of banded and periodically banded linear systems. While most of the known parallel algorithms are stable only for symmetric positive definite or diagonally dominant systems, the new algorithm incorporates pivoting without sacrificing efficiency. The principle ingredient of the algorithm is a bidiagonal cyclic reduction that admits pivoting.

We report on numerical experiments conducted on various multiprocessor computers.

1. INTRODUCTION

Methods will be discussed for the solution of linear systems

$$Ax = b \tag{1.1}$$

with n unknowns and a banded matrix A. The system matrix A has upper bandwidth k_u if $\alpha_{ij} = 0$ for $j > i + k_u$ and lower bandwidth k_l if $\alpha_{ij} = 0$ for $i > j + k_l$. If $k_u + k_l$ is small compared to n then A is said to be *narrow banded*. This is the case which shall be considered here. Frequently occurring problems include bidiagonal systems ($k_u + k_l = 1$), tridiagonal systems ($k_u + k_l = 2$) and systems obtained from the discretization of 2-dimensional partial differential equations ($k_u + k_l = \sqrt{n}$).

In addition to the ordinary banded cases the algorithms discussed here also apply to matrices which have additional nonzero elements α_{ij} when $i \leq j - n + k_l$ or $j \leq i - n + k_u$. These matrices are *periodically banded*. For example, the nonzero structure of a periodically tridiagonal matrix is

$$A = \begin{bmatrix} \times & \times & & & & \times \\ \times & \times & \times & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \times & \times & \times \\ \times & & & & \times & \times \end{bmatrix}. \tag{1.2}$$

Most of the research on parallel solvers for narrow banded systems has been concentrating on the two cases where A is either symmetric positive definite or diagonally dominant [22, 20, 24, 10, 16, 26]. In these cases Gaussian elimination is stable without pivoting. If pivoting has been included efficiency was lost [9, 32]. An alternative to Gaussian elimination with pivoting is QR factorization [2, 29]. Further articles can be found on the closely related block tridiagonal systems [5].

In the case of a *wide* band, straightforward (block) Gaussian elimination adapted to the band structure is to preferred over the algorithm presented here. In [28, 15], this situation is considered for the cases where no pivoting is needed.

The paper is organized as follows. In Section 2 the basic stable solution methods for linear systems are reviewed, in particular with respect to rounding errors and breakdown. In Section 3 techniques are introduced which enhance the parallelism and their potential danger for Gaussian elimination is discussed. The algorithms suggested for the factorization of banded linear systems are Gaussian elimination with partial pivoting applied to a permuted matrix. Thus they do not, like some other algorithms, relax the pivoting. In Section 4 the basic step of the parallel algorithms which is interpreted as a block elimination is discussed. In Section 5 we review the classical cyclic (CR) reduction algorithm for tridiagonal diagonally dominant linear systems. Section 6 discusses the simpler case of cyclic reduction for bidiagonal linear systems but includes partial pivoting. (Note that partial pivoting cannot be included into CR for the tridiagonal linear system without substantially modifying the algorithm.) In Sections 7 and 8 the new algorithm is developed for parallel banded Gaussian elimination with partial pivoting. It consists of two steps. A first step, which is discussed in Section 7 reduces the large banded matrix to a smaller block bidiagonal system. This step does not involve any communication. The second step implements a block bidiagonal elimination CR algorithm with partial pivoting. In Section 9 the parallel speedup of this algorithm compared to the corresponding one-processor LAPACK subroutine is discussed.

2. Basic Methods for the solution of linear systems of equations

Two main methods for the direct solution of general linear systems are implemented in modern software packages like LAPACK [1]. The first method is Gaussian elimination with partial pivoting and the second is based on QR factorization. Other algorithms are used if the matrix A has special properties. Examples are Cholesky factorization for symmetric positive A and the Bunch-Kaufmann-Parlett algorithm for symmetric indefinite matrices. See [13] for a comprehensive overview on the direct solution of linear systems of equations. In the following sections the case of banded linear systems will be considered. The corresponding algorithms use the fact that many elements of banded matrices are zero and thus do not need to be neither stored nor processed. The algorithms which will be discussed can be viewed as special cases of algorithms used to solve general sparse linear systems [11].

In the course of Gaussian elimination with partial pivoting a permutation matrix P, a unit upper triangular matrix L and a lower triangular matrix U are computed such that

$$PA = LU, \tag{2.1}$$

where A is the system matrix. Given this factorization the linear system $Ax = b$ is solved in two steps: First $Ly = Pb$ is solved by forward elimination and then $Ux = y$ is solved by back-substitution using the fact that both L and U are triangular.

All the computations are assumed to be done in floating point arithmetic. Thus rounding errors have to be taken into account. The precision of the computations is influenced by

- the condition of the matrix A and
- the stability of the algorithm used.

While for a given problem one has no influence on the condition of A the choice of a stable algorithm is essential. For example, if no pivoting is used very large rounding

errors can occur and, in some cases, the LU factorization does not even exist, a fact which is termed *break-down*. Breakdown can happen for perfectly conditioned matrices and is by no means unusual. An example of a matrix which does not have an LU-factorization is

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.2}$$

The effect of rounding shall now be discussed in more detail. First, assume the case of "ideal arithmetic", i.e., all the computations can be done with no rounding errors but the initial data A and b are rounded. Thus, the data used in the computations is $A + E$ and $b + e$ where the elements of E and e are all of size of the machine precision ϵ times the exact values. As a consequence, "near-by" equations

$$(A + E)\hat{x} = b + e \tag{2.3}$$

are solved in place of $Ax = b$. Now if E and e are "small enough" then one gets [13, p.80]

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \kappa(A) \left(\frac{\|e\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right) + O(\epsilon^2) \tag{2.4}$$

for the relative error. Here $\|\cdot\|$ denotes any vector norm and $\kappa(A) = \|A\|\|A^{-1}\|$ is the corresponding condition number of A.

The effect of rounding on the elimination process is analyzed by *backward error analysis*. It can be seen [31] that the computed solution $\hat{x}$ of $Ax = b$ using Gaussian elimination with partial pivoting and floating point arithmetic satisfies in exact arithmetic the equation

$$(A + E)\hat{x} = b \tag{2.5}$$

where E is bounded by

$$\|E\|_\infty \leq n\epsilon(\|A\|_\infty + 5n\|\hat{U}\|_\infty) + O(\epsilon^2). \tag{2.6}$$

A commonly used bound which does not involve $\hat{U}$ explicitly is

$$\|E\|_\infty \leq 8n^3\rho\|A\|_\infty\,\epsilon + O(\epsilon^2) \tag{2.7}$$

where the *growth factor* ρ may depend on A (and n). Using this bound and estimate (2.4) one gets

$$\frac{\|\hat{x} - x\|}{\|x\|} \leq \kappa_\infty(A) 8n^3\rho\,\epsilon + O(\epsilon^2). \tag{2.8}$$

It is known that $\rho \leq 2^n$. Wilkinson [31] has given an example of a matrix A where $\rho = 2^n$ occurs but it is often thought that this example is contrived and fast growth of ρ is not seen in practice. However, it will be shown later that there are classes of practically relevant problems which do show an exponential growth of ρ, in particular for parallel algorithms. Thus, if Gaussian elimination with partial pivoting is used, the size of the elements of U should be monitored during factorization as it could be necessary to abandon the computations because of problems with rounding errors.

An alternative to Gaussian elimination is based on QR factorization and uses Householder reflectors. It involves computation of an orthogonal matrix Q (possibly in factorized form) and an upper triangular matrix R such that

$$A = QR \tag{2.9}$$

holds. Given this factorization the solution of $Ax = b$ is obtained from the system

$$Rx = Q^T b \tag{2.10}$$

which is solved by back-substitution. It is shown in [23, p.92] that if the factorization is done in floating point arithmetic with precision ϵ the so obtained solution $\hat{x}$ solves the equation

$$(A + E)\hat{x} = b + e. \tag{2.11}$$

and the matrix E and the vector e are bounded by

$$\|E\|_F \leq (3n^2 + 41n)\|A\|_F\, \epsilon + O(\epsilon^2), \tag{2.12}$$

$$\|e\| \leq (3n^2 + 40n)\|b\|\, \epsilon + O(\epsilon^2). \tag{2.13}$$

Instead of the maximum norm the Frobenius norm $\|\cdot\|_F$ is used here. While the Frobenius norm can be much larger than the maximum norm these bounds only have a factor n^2 and, more importantly, they do not involve a (potentially exponentially increasing) growth factor ρ. This is why QR factorization is thought to be much more stable than Gaussian elimination even if partial pivoting is used. The trade-off are the higher factorization costs.

With the earlier bound in (2.4) one gets from this an estimate for the error of the floating point solution in the 2-norm:

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq \kappa_2(A)(3n^2 + 41n)(1 + \frac{\|A\|_F}{\|A\|_2})\, \epsilon + O(\epsilon^2). \tag{2.14}$$

This bound and, in fact, this algorithm is only useful for well conditioned systems. In the case of ill-conditioned systems and, in particular, singular systems, column pivoting has to be used [13].

These methods have been implemented in parallel if the bandwidth of A is large enough (see, e.g., [6]). However, in this paper narrow banded matrices are considered. For narrow banded matrices the amount of parallelism of both LU factorization and QR factorization is too small for massively parallel computers and both the LU factorization and QR factorization are essentially recursive. Many alternatives have been suggested to solve banded linear systems in parallel, see [4] for a review of some of them. However, the applicability of these parallel solvers is usually restricted to the case of positive definite and diagonally dominant matrices. In particular, they can suffer from breakdown if applied to a general linear system. It is probably less well known that they may also show some severe instabilities relating to exponential growth of the elements of the factor matrices even for practically relevant problems (see Section 3). This is in contrast to the methods in LAPACK which do not break down and in the case of Gaussian elimination rarely are unstable, in the case of QR factorization never are unstable.

Alternative algorithms with similar stability properties as ours have been suggested by [30, 17, 32, 12].

3. Factorizations of permuted systems

In the following new methods will be described which have the same basic properties as the methods implemented in LAPACK but in addition allow an efficient implementation on MIMD distributed memory parallel computers. The key idea of all these methods is to modify the order of elimination (i.e., the numbering of the

unknowns) in order to increase parallelism. Thus, instead of solving $Ax = b$, an equivalent system $A'x' = b'$ is solved such that Gaussian elimination with partial pivoting or QR factorization displays a high degree of parallelism. The choices of the equivalent system involves permutations and typically

$$A' = AS, \qquad b' = b, \tag{3.1}$$

where S is a permutation matrix. In the following suggestions how to choose S will be made. But first, the effect of permutations on the factorization methods is discussed.

Thus, for the parallel algorithms, the following factorization replaces the ordinary LU factorization of equation (2.1):

$$PAS = LU. \tag{3.2}$$

This corresponds to Gaussian elimination with partial pivoting where P, L and U are as before. Factorizations of this kind have earlier been used to decrease fill-in for sparse factorization [11]. Another instance of this kind of factorization occurs with complete pivoting. Application of complete pivoting can reduce the growth factor substantially. So, in a sense, the freedom one has in choosing S can either be used to get faster algorithms as in the case of fill-reducing and parallelism-increasing reorderings or to enhance stability as in the case of complete pivoting. These two goals seem to contradict each other, however, compromises are feasible. Using the above factorization the solution of $Ax = b$ is obtained in three steps: First $Lu = b$ is solved by forward elimination, then $Uz = y$ is solved by back-substitution and finally the permutation $x = Sz$ is performed.

The permutation does not change the conditioning of the problem, i.e., the system $A'x' = b'$ has the same condition as $Ax = b$ if $A' = AS$. However, and this might be less well-known, it can have a large effect on the growth factor ρ even for practically important problems. An example can be derived from an example given in [34] and shall be discussed in the following.

Assume that one wishes to solve the initial value problem

$$y'(t) = My(t), \quad y(0) = y_0, \quad M = \begin{bmatrix} -1/6 & 1 \\ 1 & -1/6 \end{bmatrix}. \tag{3.3}$$

One way of solving this is with multiple shooting [27]. If an exact integrator is used on each of the subintervals one obtains a linear system $Ax = b$ with a matrix A of the form

$$A = \begin{bmatrix} I & & & & \\ -e^{Mh} & I & & & \\ & -e^{Mh} & I & & \\ & & \ddots & \ddots & \\ & & & -e^{Mh} & I \end{bmatrix}. \tag{3.4}$$

Of course, in practice, the integrator on the subintervals would not be exact. However, the matrix elements would be arbitrarily close to the ones in the example above, with e^{Mh} replaced by a rational approximation.

In multistep methods h is chosen to be small. In particular, assume that h is such that

$$e^{Mh} = I + Mh + O(h^2) = \begin{bmatrix} 1 - h/6 & h \\ h & 1 - h/6 \end{bmatrix} + O(h^2) \tag{3.5}$$

has only elements which are less than 1. Then the application of Gaussian elimination with partial pivoting yields $P = I$, $L = A$ and $U = I$.

Now chose S such that the first block column is moved to the end. (The permutations chosen for the parallel algorithms are very similar and typically move intermediate blocks to the end.) Then

$$AS = \begin{bmatrix} 0 & & & & I \\ I & 0 & & & -e^{Mh} \\ -e^{Mh} & I & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & -e^{Mh} & I & 0 \end{bmatrix}. \tag{3.6}$$

Because of the restrictions on h one gets $P = S^T$ in this case for Gaussian elimination with partial pivoting and so

$$\begin{aligned} S^T AS &= \begin{bmatrix} I & & & & -e^{Mh} \\ -e^{Mh} & I & & & \\ & \ddots & \ddots & & \\ & & -e^{Mh} & I & \\ & & & 0 & I \end{bmatrix} \\ &= \begin{bmatrix} I & & & & \\ -e^{Mh} & I & & & \\ & \ddots & \ddots & & \\ & & -e^{Mh} & I & \\ & & & 0 & I \end{bmatrix} \begin{bmatrix} I & & & & -e^{Mh} \\ & I & & & -e^{2Mh} \\ & & \ddots & & \vdots \\ & & & I & -e^{(n-1)Mh} \\ & & & & I \end{bmatrix} = LU. \end{aligned}$$

While h can be very small, n can be very large independently of h. Thus, the factor U and, by consequence, the rounding errors can be arbitrarily large. The growth factor will be of the order of $e^{n\lambda_1 h}$ where λ_1 is the largest eigenvalue of M. While in the case of parallel execution the growth factor is some root of this it can still be extremely large, in particular for a small number of processors. So, in this case, it is advisable to use a more stable method and Wright [33] suggests to use QR factorization. As this is usually at least twice as expensive, it is suggested that typically LU factorization with partial pivoting should be tried and the growth factor should be monitored. If this factor turns out to be too large then one should switch to QR factorization.

The QR factorization related to the permuted problem is

$$AS = QR. \tag{3.7}$$

Once this factorization is established, $Ax = b$ is solved by first solving $Rz = Q^T b$ and then permuting such that $x = S^T z$. Note that for QR factorization the permutation S does have no influence on the error bounds in equation (2.14). Further note that this method is not suitable for singular or ill-conditioned A as in this case column pivoting is required. This would interfere with the permutation S and might destroy

parallelism. For singular matrices different methods are required; in contrast, QR factorization is suggested here as a very stable method for parallel solvers for very well conditioned systems.

4. Block elimination schemes

Given factorizations (3.2) and (3.7) the linear system $Ax = b$ is readily solved using permutations and solvers for triangular and orthogonal systems. Of course the factorizations have to be such that not only can they be computed in parallel but also the permutations and solution of triangular and orthogonal systems need to be computed in parallel as well, and, in particular, scalability has to be guaranteed. Note that the choice of S is not unique, and, in fact, by varying S different algorithms are obtained. In the previous section it was discussed what effect such permutations can have on the stability of the overall algorithm.

In practice, the permutations S are given as a sequence of simple permutations (like the odd/even sort permutation). Thus

$$S = S_1 S_2 \cdots S_t. \tag{4.1}$$

Corresponding to this representation of S the algorithm proceeds in t parallel steps. Each of these steps eliminates a consecutive set of unknowns. An elimination step which corresponds to the elimination of a set of unknowns will be called *block elimination*. It has also been called reduction earlier.

Consider block elimination relating to Gaussian elimination with partial pivoting. This generates a factorization of the form

$$PAS = \begin{bmatrix} L & \\ F & I \end{bmatrix} \begin{bmatrix} U & E \\ & A^{(1)} \end{bmatrix}. \tag{4.2}$$

This is the part of the LU factorization corresponding to the elimination of a first set of unknowns. The matrix L is unit lower triangular, U is upper triangular, I is the identity and E and F are general rectangular matrices. The matrix $A^{(1)}$ is often called *Schur complement* or matrix of the *reduced system*. As before, P is the permutation matrix which is required for partial pivoting and S is the permutation matrix used to increase the amount of parallelism.

A block elimination step using QR factorization corresponds to the factorization

$$AS = Q_1 \begin{bmatrix} R_1 & G_1 \\ & A_1 \end{bmatrix}. \tag{4.3}$$

In this case Q_1 is orthogonal, R_1 is upper triangular and G_1 is a general matrix. The matrix A_1 is in general not the same as the matrix $A^{(1)}$ of the Gaussian elimination case.

Similar ideas are used in the case of Cholesky factorization and LU factorization for diagonally dominant linear systems. In the case of Cholesky factorization one gets

$$S^T AS = \begin{bmatrix} U^T & \\ E^T & I \end{bmatrix} \begin{bmatrix} U & E \\ & A^{(1)} \end{bmatrix} \tag{4.4}$$

and in the case of LU factorization

$$S^T AS = \begin{bmatrix} L & \\ F & I \end{bmatrix} \begin{bmatrix} U & E \\ & A^{(1)} \end{bmatrix}. \tag{4.5}$$

Algorithms based on these factorizations include divide and conquer but also cyclic reduction and generalizations thereof. The main difference between the new algorithms for LU with partial pivoting and QR factorization and the older ones for Cholesky and LU factorization without pivoting is that the older algorithms use symmetric permutations of the form $A \mapsto S^T A S$ to gain more parallelism whereas the new algorithms use one-sided permutations of the form $A \mapsto AS$.

In the next two sections the simplest cases of banded linear systems are discussed.

5. Cyclic reduction for tridiagonal diagonally dominant systems

This section gives a review on a historic idea and is intended as a motivation for the following. Cyclic reduction (CR) has been introduced 1965 by Hockney in collaboration with Golub [18, 8] and by Buneman [7]. Many authors have discussed variations and implementations of it, mainly in connection with fast Poisson solvers.

CR has been used to solve systems with column diagonally dominant tridiagonal matrices

$$A = \begin{bmatrix} \alpha_0 & \gamma_0 & & & \\ \beta_1 & \alpha_1 & \gamma_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-2} & \gamma_{n-2} \\ & & & \beta_{n-1} & \alpha_{n-1} \end{bmatrix}, \qquad \text{where} \quad |\alpha_k| \geq |\beta_{k+1}| + |\gamma_{k-1}|. \tag{5.1}$$

The block elimination steps of cyclic reduction eliminate first half of all the unknowns, then half of the remaining unknowns etc. The permutation S used for one block elimination step is the odd-even sort permutation, with the property

$$[\xi_0, \xi_1, \xi_2, \xi_3, \dots]\, S = [\xi_0, \xi_2, \dots \mid \xi_1, \xi_3, \dots] \tag{5.2}$$

for any vector $[\xi_0, \xi_1, \xi_2, \xi_3, \dots]$.

The symmetrically permuted matrix thus has the form

$$S^T A S = \left[\begin{array}{ccc|ccc} \alpha_0 & & & \gamma_0 & & \\ & \alpha_2 & & \beta_2 & \gamma_2 & \\ & & \ddots & & \ddots & \ddots \\ \hline \beta_1 & \gamma_1 & & \alpha_1 & & \\ & \beta_3 & \ddots & & \alpha_3 & \\ & & \ddots & & & \ddots \end{array}\right]. \tag{5.3}$$

The two diagonal blocks of $S^T A S$ are diagonal, the two off-diagonal blocks are bidiagonal matrices.

From this a partial LU factorization is obtained as

$$S^T A S = \left[\begin{array}{ccc|ccc} 1 & & & 0 & & \\ & \ddots & & & \ddots & \\ & & 1 & & & 0 \\ \hline \lambda_1 & \mu_1 & & 1 & & \\ & \lambda_3 & \ddots & & \ddots & \\ & & \ddots & & & 1 \end{array}\right] \left[\begin{array}{ccc|ccc} \alpha_0 & & & \gamma_0 & & \\ & \alpha_2 & & \beta_2 & \gamma_2 & \\ & & \ddots & & \ddots & \ddots \\ \hline 0 & & & \alpha_0^{(1)} & \gamma_0^{(1)} & \\ & \ddots & & \beta_1^{(1)} & \alpha_1^{(1)} & \ddots \\ & & 0 & & \ddots & \ddots \end{array}\right]. \tag{5.4}$$

This is a block LU factorization as the upper left block of the left factor is equal to I. Furthermore, the Schur complement of this block is again tridiagonal. The factorization generalizes to block tridiagonal and banded systems. However existence of all these factorizations is not guaranteed in general; only for special cases like symmetric positive definite or diagonally dominant matrices they are known to exist.

The components of these factors are computed as

$$\left.\begin{aligned} \lambda_{2i+1} &= \beta_{2i+1}/\alpha_{2i}, \\ \mu_{2i+1} &= \gamma_{2i+1}/\alpha_{2i+2}, \\ \gamma_i^{(1)} &= -\mu_{2i+1}\gamma_{2i+2}, \\ \beta_i^{(1)} &= -\lambda_{2i+1}\beta_{2i}, \\ \alpha_i^{(1)} &= \alpha_{2i+1} - \lambda_{2i+1}\gamma_{2i} - \mu_{2i+1}\beta_{2i+2}. \end{aligned}\right\} \quad i = 1, \ldots, \lfloor \tfrac{n}{2} \rfloor - 1, \tag{5.5}$$

These formulas are directly obtained from the factorization. Consequently, $8n + O(1)$ operations are required to complete the factorization. Note that in (5.5) the computations for different indices i are independent of each other. To store the factorization computed by CR $5n$ memory locations are needed. Here, we assume that λ_i and μ_i overwrite β_i and γ_i, respectively.

The LU factorization of the unpermuted matrix $A = LU$ is given by

$$\left.\begin{aligned} \delta_0 &= \alpha_0, \\ \lambda_i &= \beta_i/\delta_{i-1}, \\ \delta_i &= \alpha_i - \lambda_i\gamma_{i-1}, \end{aligned}\right\} \quad i = 1, \ldots, n-1.$$

Here, only $3(n-1)$ operations are required. However, this is essentially a recursion. With δ_i and λ_i overwriting α_i and β_i, respectively, the LU factorization needs only $3n$ memory locations.

The factorization by cyclic reduction requires about 2.7 times as many operations as the ordinary LU factorization. It is said that cyclic reduction has a *redundancy* of 2.7 [19]. The number of floating point operations however is a poor measure of performance for parallel computers as the level of parallelism is not taken into account. Cyclic reduction in particular was introduced in order to increase the level of parallelism to achieve better performance on parallel computers. The *parallel complexity* t_{opt} of an algorithm is the minimal time it takes to run this algorithm on a parallel computer with an arbitrary number of processors [19]. No matter how many processors are available, the algorithm will never run in a shorter time than t_{opt}. The ratio of t_{opt} and the complexity of the "best" sequential algorithm is called *parallel speedup* s_{opt}.

For the analysis of cyclic reduction it will be assumed that n and the number of processors p are powers of 2 and that all the four floating point operations take the same amount of time t_{F}. First, the parallel speedup shall be estimated without taking communication cost into account, i.e., for shared memory computers. The bound which is obtained in this way is fundamental. Speedups beyond it are impossible.

It can be seen from (5.5) that in one sweep of CR first all the $n/2$ divisions and then all $2n$ multiplications can be done in parallel. Finally, two time units are needed for twice $n/2$ synchronous subtractions. Thus, if $2n$ processors are available, the parallel complexity of CR is

$$t_{\text{opt}} = 4\log_2(n)t_{\text{F}}. \tag{5.6}$$

Comparing with LU factorization, the best sequential algorithm, the parallel speedup becomes

$$s_{\text{opt}} = 3n/4\log_2(n). \tag{5.7}$$

So, on an arbitrary large parallel computer a problem of size $n = 1024$ cannot be solved more than 76.8 times faster than on a single processor of this computer. The optimal speedup is achieved with 2048 processors. While the parallel speedup is the optimum with respect to the number of processors it does depend on the problem size. If, for example, $n = 2^{20} = 1,048,576$ then the parallel speedup is almost 40,000.

In practice, most parallel computers with a large number of processors have distributed memories. The time to access n non-local data items is commonly modelled in the form $\sigma + \tau n$ where σ is the communication startup time and τ is the reciprocal of the bandwidth of the communication network. We set $\sigma = \mu t_{\text{F}}$ where μ on a typical multicomputer is at least 1000. As τ is much smaller than μt_{F} the communication of short messages is completely dominated by the startup time.

If communication is taken into account the execution time of cyclic reduction does not decrease monotonically with the number p of processors. There is a number p which yields the lowest parallel complexity and thus highest speedup. If p processors are available then the first $\log_2(n/p)$ steps of CR do not require any communication. The time for these steps is approximately $8(n-p)/p\,t_{\text{F}}$. The last $\log_2(p)$ steps do require communication. If we assume that communication completely dominates the computation the time for these steps is approximately $\mu\log_2(p)\,t_{\text{F}}$. Thus, the total time is

$$t \approx (8(n-p)/p + \mu\log_2(p))\,t_{\text{F}}. \tag{5.8}$$

t attains its minimum at $p = 8\log(2)\,n/\mu \approx 5.5\,n/\mu$. Thus, the parallel complexity is

$$t_{\text{opt}} \approx (\mu/\log(2))(1+\log(8\log(2)\,n/\mu))t_{\text{F}} \approx (1.4+\log_2(5.5\,n/\mu))\;\mu t_{\text{F}}. \tag{5.9}$$

Therefore, the parallel speedup for distributed memory computers becomes

$$s_{\text{opt}} \approx 3n/(\mu(1.4+\log_2(5.5\,n/\mu)))\,. \tag{5.10}$$

In the case of $\mu = 1000$ and $n = 1024$ one obtains a parallel "speedup" $s_{\text{opt}} \approx 0.76$ with $p = 8$. The high latency prohibits a speedup for such a small problem size. In the case of $n = 2^{20}$ the upper bound for speedup, i.e. the parallel speedup, is around 225. It is obtained with 8192 processors. So, the parallel speedup drops by about a factor of 180 when moving from a shared memory multiprocessor to a distributed memory multicomputer.

Thus, it can be concluded that parallel processing has some hard limitations, in particular if communication is involved. Nevertheless, it is possible to achieve good speedups, i.e. speedups close to the number of processors divided by the redundancy, if the problem size is large enough and the number of processors is not too large. This adaptation of problem size to processor number is related to scalability. Algorithms are called scalable, if for any number of processors there are problem sizes permitting good speedups. In other words, algorithms are scalable if the ration between speedup and processor number (efficiency) can be held fixed as the processor number increases if only the problem size grows sufficiently quickly [14].

For example, equation (5.8) implies that the speedup $s = (3nt_{\mathrm{F}})/t$ obtained when solving a problem of size $n = 2^{20}$ with $p = 16$ processors is very close to $3p/8 = 6$, the limit given by the redundancy. Because of the increasing influence of communication, for $p = 128$ one only gets a speedup of 24 which is half of $3p/8$. So, a problem of size $n = 2^{20}$ has to be considered small for a computer with 128 processors.

Similar remarks hold for Cholesky factorization vs. symmetric CR. It will be shown that a specific variant of cyclic reduction can be developed for tridiagonal systems and Gaussian elimination with partial pivoting or QR factorization. As the factorizations are not as straight-forward as in the case of LU or Cholesky factorization the simpler case of bidiagonal systems will be treated first.

6. Cyclic reduction for bidiagonal systems using LU factorization with partial pivoting

In the following a parallel algorithm for the factorization of lower bidiagonal systems with periodic nonzero structure is established. The same algorithm can be used to solve ordinary lower bidiagonal systems and, with a minor modification, to solve upper bidiagonal systems. A review of parallel solvers for bidiagonal systems can be found in [21].

Ordinary bidiagonal systems are just linear recursions which could be solved by substitution. This has two disadvantages: First, the recursion could be unstable and lead to incorrect results, and second, the recursive nature does inhibit parallel execution. The instability of the recursion is addressed by partial pivoting. As in the case of tridiagonal systems column permutations are used to enhance parallelism.

The matrix to be factored is of the form

$$A = \begin{bmatrix} \alpha_0 & & & \beta_0 \\ \beta_1 & \alpha_1 & & \\ & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} \end{bmatrix}. \tag{6.1}$$

The permutation S used for the cyclic reduction algorithm is the same as in the tridiagonal case, i.e., odd-even sort. Applying S to A from the right, one gets

$$AS = \left[\begin{array}{cccc|cccc} \alpha_0 & & & & & & & \beta_0 \\ \beta_1 & & & & \alpha_1 & & & \\ & \alpha_2 & & & \beta_2 & & & \\ & \beta_3 & & & & \alpha_3 & & \\ & & \alpha_4 & & & \beta_4 & & \\ & & & \ddots & & & \ddots & \end{array}\right]. \tag{6.2}$$

This permutation causes the even unknowns to be eliminated first. In this case the pivoting of the first block elimination step is described by exponents $\epsilon_k \in \{0, 1\}$

in the factorization

$$S^T \begin{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^{\epsilon_0} & & \\ & \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^{\epsilon_1} & \\ & & \ddots \end{bmatrix} AS = \tag{6.3}$$

$$\left[\begin{array}{ccc|ccc} 1 & & & 0 & & \\ & \ddots & & & \ddots & \\ & & 1 & & & 0 \\ \hline \lambda_1 & & & 1 & & \\ & \lambda_2 & & & \ddots & \\ & & \ddots & & & 1 \end{array}\right] \left[\begin{array}{ccc|ccc} \omega_0 & & & \gamma_0 & & \delta_0 \\ & \omega_1 & & \delta_1 & \gamma_1 & \\ & & \ddots & & \ddots & \ddots \\ \hline 0 & & & \alpha_0^{(1)} & & \beta_0^{(1)} \\ & \ddots & & \beta_1^{(1)} & \alpha_1^{(1)} & \\ & & 0 & & \ddots & \ddots \end{array}\right].$$

All the components of this factorization are obtained from the $n/2$ independent order 2 problems. Each problem corresponds to a factorization of a 2×3 matrix using Gaussian elimination with partial pivoting:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^{\epsilon_i} \begin{bmatrix} \alpha_{2i} & 0 & \beta_{2i} \\ \beta_{2i+1} & \alpha_{2i+1} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \lambda_i & 1 \end{bmatrix} \begin{bmatrix} \omega_i & \delta_i & \gamma_i \\ 0 & \beta_i^{(1)} & \alpha_i^{(1)} \end{bmatrix}, \quad 0 \le i < n/2. \tag{6.4}$$

From this the formulas for the coefficients are easily obtained. The actual operations depend on the exponent ϵ_i, i.e., on the pivoting. In any case, computation is only required for the λ_i, the $\beta_i^{(1)}$ and the $\alpha_i^{(1)}$. In the case without row interchanges ($\epsilon_i = 0$) a division is needed for λ_i, no operation for $\beta_i^{(1)}$, and a multiplication for $\alpha_i^{(1)}$. In the case of $\epsilon_i = 1$, i.e. when rows are interchanged, λ_i requires a division, $\beta_i^{(1)}$ a multiplication while $\alpha_i^{(1)}$ is free. So, irrespective of pivoting, the factorization (6.3) costs $n/2$ multiplications and $n/2$ divisions. Note that the Schur complement $A^{(1)}$ is an $n/2$ by $n/2$ (periodic) bidiagonal matrix and so the scheme can be applied recursively. The overall complexity is thus $2n$ floating point operations.

The LU factorization of the unpermuted A in (6.1) costs the same number of floating point operations as the last column fills up, unless $\beta_0 = 0$. Thus, there are no redundant computations and the sequential execution of both algorithms lasts

$$t = 2nt_F. \tag{6.5}$$

Both, the sequential and parallel algorithm uses $4n$ memory locations to store the factorization.

The parallel complexity is obtained by a similar analysis as in the previous section. While all the order 2 factorizations can be executed simultaneously, λ_i has to be computed before $\alpha_i^{(1)}$ or $\beta_i^{(1)}$. Therefore, under the same simplifying assumptions as earlier, the parallel complexity is

$$t_{\text{opt}} = \log_2(n) 2t_F \tag{6.6}$$

at the expense of only $n/2$ processors. The parallel speedup is

$$s_{\text{opt}} = n/\log_2(n). \tag{6.7}$$

This is up to a factor 3/4 the same as for tridiagonal CR, cf. (5.7). Thus, similar remarks hold. However, only a quarter of the processors is needed to get the same speedup.

If the cost of communication is included in the considerations we have

$$t \approx (2(n-p)/p + m\log_2(p))t_F. \tag{6.8}$$

The minimum is in $p = 2\log(2)n/m \approx 1.4n/m$ and so the parallel complexity is

$$t_{\text{opt}} = m(1.4 + \log_2(1.4n/m))t_F \tag{6.9}$$

and the parallel speedup is

$$s_{\text{opt}} = 2n/(m(1 + \log_2(2n/m))). \tag{6.10}$$

In the case of $n = 1024$ and $m = 1000$ one gets a parallel speedup $s_{\text{opt}} = 1.05$ with 2 processors. In the case $n = 2^{20}$ on gets $s_{\text{opt}} \approx 174.4$ with $p = 2024$. This is less than for the tridiagonal CR because of the higher ratio of communication to computation volume.

The scalability is very similar to the case of CR for tridiagonal systems. For example, if $p = 16$ and $n = 2^{20}$ one gets a speedup very close to p.

Very similar formulas can be obtained if in (6.4) QR factorization with Givens rotations is used instead of Gaussian elimination. In this case one gets a parallel factorization of the form

(6.11)

$$AS = \begin{bmatrix} \begin{bmatrix} \gamma_1 & \sigma_1 \\ -\sigma_1 & \gamma_1 \end{bmatrix} & & \\ & \begin{bmatrix} \gamma_2 & \sigma_2 \\ -\sigma_2 & \gamma_2 \end{bmatrix} & \\ & & \ddots \end{bmatrix} S \left[\begin{array}{ccc|ccc} \omega_0 & & & \gamma_0 & & \\ & \omega_1 & & \delta_1 & \gamma_1 & \\ & & \ddots & & \ddots & \ddots \\ \hline 0 & & & \alpha_0^{(1)} & & \beta_0^{(1)} \\ & \ddots & & \beta_1^{(1)} & \alpha_1^{(1)} & \\ & & 0 & & \ddots & \ddots \end{array}\right].$$

The components are again computed from $n/2$ independent problems,

(6.12)

$$\begin{bmatrix} \alpha_{2i} & 0 & \beta_{2i} \\ \beta_{2i+1} & \alpha_{2i+1} & 0 \end{bmatrix} = \begin{bmatrix} \gamma_i & -\sigma_i \\ \sigma_i & \gamma_i \end{bmatrix} \begin{bmatrix} \omega_i & \gamma_i & \delta_i \\ 0 & \alpha_i^{(1)} & \beta_i^{(1)} \end{bmatrix}, \qquad i = 0, \ldots, n/2-1.$$

The first factor on the right-hand side is orthogonal and so one has

$$\gamma_i^2 + \sigma_i^2 = 1.$$

Similar remarks as before apply. In particular, the reduced system is again periodically lower bidiagonal. The overall complexity of this QR factorization is $9n$ flops, i.e. 4.5 times as much as Gaussian elimination.

These factorizations are generalized to block bidiagonal systems. Note that by applying a right shift every banded matrix is transformed into a periodically lower banded matrix. If this matrix is partitioned one automatically obtains a block bidiagonal matrix such that the block bidiagonal cyclic reduction idea can be applied to this matrix.

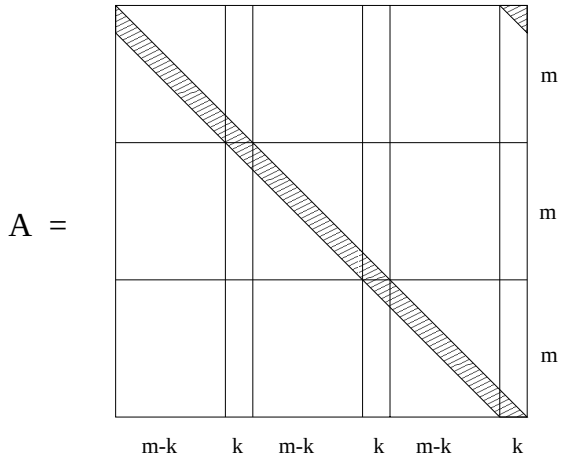

FIGURE 7.1. Partitioning of a banded matrix A

7. The first block elimination step for parallel band solvers

The parallel band solver discussed in the sequel consists of a succession of steps which can be interpreted as LU factorizations of the form defined in (4.2). While the second and later steps all factorize block bidiagonal matrices the first step is different in that it takes into account the banded structure (which is lost in the later steps).

The parallel factorizations are largely simplified if A is in *standard form* which shall mean that $k_u = 0$, i.e., that A is a lower banded matrix. For any periodically banded matrix a circular shift S_0 is found such that S_0A (or, equivalently, AS_0) is in standard form. As a consequence, for every banded matrix there is a permutation S_0 such that S_0A and AS_0 are in standard form. Note that the standard form is not uniquely defined. For an ordinary tridiagonal matrix, e.g., it can be

$$\begin{bmatrix} 0 & & & & \times & \times \\ \times & \times & & & & \\ \times & \times & \times & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \times & \times & \times \end{bmatrix} = S_0A$$

or

$$\begin{bmatrix} \times & & & & & \times \\ \times & \times & & & & \times \\ \times & \times & \times & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \times & \\ & & & \times & \times & 0 \end{bmatrix} = AS_0.$$

In general, if A is in standard form then so is S^TAS for any cyclic shift S. In the following it will be assumed that the banded matrix is in *standard form* with lower bandwidth k. This is the standard form best suited for both LU factorization with partial pivoting and QR factorization.

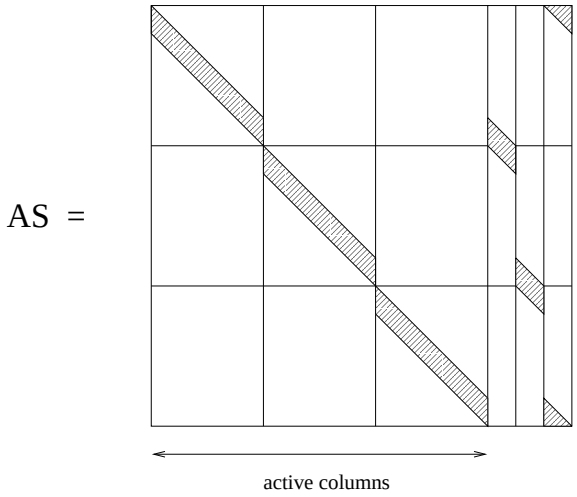

FIGURE 7.2. Permutation AS of matrix A from Figure 7.1

The permutation S and the block elimination are chosen based on a distribution of the rows of A to p processors of a MIMD computer with distributed memory. Assume, for simplicity of illustration, that $n = pm$. Then the set of rows is partitioned into p equally sized sets. The columns in turn are partitioned in blocks of size $(m-k, k, m-k, k, m-k, \dots)$. In the case $p=3$ this leads to a blocking of A as displayed in Fig. 7.1.

In general the partitioned matrix has the following basic structure,

$$A = \begin{bmatrix} A_{11} & A_{12} & & & & & & & A_{1p} \\ & A_{21} & A_{22} & A_{23} & & & & & \\ & & & A_{32} & A_{33} & A_{34} & & & \\ & & & & & & \ddots & & \\ & & & & & & & A_{p,p-1} & A_{pp} & A_{p,p+1} \end{bmatrix}. \tag{7.1}$$

Note in particular that this is not a periodic block tridiagonal structure. The blocks $A_{i,i-1}$ and $A_{i,i+1}$ are both m by k matrices and the blocks $A_{i,i}$ are m by $m-k$ matrices. The block columns of width k are called *separators*. Here, the width of the separators equals the bandwidth k. In the case of banded diagonally dominant or symmetric positive definite matrices it suffices to chose the width of the separators equal to the larger of the half-bandwidths k_l and k_u [20, 4]. As the bandwidth is often twice the half-bandwidth, our approach is called double-width separator approach [9, 25].

The block elimination scheme will eliminate the unknowns corresponding to the columns in the blocks of size m by $m-k$. In the case $p=3$ this corresponds to a permutation S the effect of which is displayed in Fig. 7.2, see also (6.2). The *active columns* are the columns corresponding to the unknowns eliminated in the block elimination step. The important point to note here is that all columns in the active set have nonzero elements only in one partition, i.e., on one processor. Furthermore all the rows corresponding to these nonzero elements are on the same processor as well as the partitioning is row-based. Thus, the block elimination step corresponding to the active set consists of p independent block eliminations, one for each processor. This holds for both Gaussian elimination with partial pivoting and QR factorization.

In the first step of Gaussian elimination the diagonal m by $m-k$ blocks are factorized as

$$P_i A_{ii} = \begin{pmatrix} L_i & 0 \\ F_i & I_{k\times k} \end{pmatrix} \begin{pmatrix} U_i \\ 0 \end{pmatrix} = \begin{pmatrix} L_i \\ F_i \end{pmatrix} U_i, \quad i = 1, \ldots, p, \tag{7.2}$$

where U_i and L_i are triangular $m-k$ by $m-k$ matrices and F_i is a k by $m-k$ matrix. The off-diagonal blocks are then multiplied with P_i and the inverse of the "L-factor" of A_{ii} to get

$$\begin{pmatrix} E_{i,i-1} \\ A^{(1)}_{i,i-1} \end{pmatrix} := \begin{pmatrix} L_i & \\ F_i & I \end{pmatrix}^{-1} P_i A_{i,i-1} = \begin{pmatrix} L_i^{-1} & \\ -F_i L_i^{-1} & I \end{pmatrix} P_i A_{i,i-1}, \tag{7.3}$$

$$\begin{pmatrix} E_{i,i} \\ A^{(1)}_{i,i} \end{pmatrix} := \begin{pmatrix} L_i & \\ F_i & I \end{pmatrix}^{-1} P_i A_{i,i+1} = \begin{pmatrix} L_i^{-1} & \\ -F_i L_i^{-1} & I \end{pmatrix} P_i A_{i,i+1}. \tag{7.4}$$

From this, the matrices P, L, F and U of equation (4.2) are block diagonal matrices with diagonal blocks P_i, L_i, F_i, and U_i, respectively. The other two matrices in equation (4.2) are

$$E = \begin{bmatrix} E_{11} & & & E_{1p} \\ E_{21} & E_{22} & & \\ & \ddots & \ddots & \\ & & E_{p,p-1} & E_{pp} \end{bmatrix} \tag{7.5}$$

and

$$A^{(1)} = \begin{bmatrix} A^{(1)}_{11} & & & A^{(1)}_{1p} \\ A^{(1)}_{21} & A^{(1)}_{22} & & \\ & \ddots & \ddots & \\ & & A^{(1)}_{p,p-1} & A^{(1)}_{pp} \end{bmatrix} \tag{7.6}$$

Corresponding formulas are valid for the QR factorization where $\left(\begin{smallmatrix} L_i \\ F_i \end{smallmatrix}\ I\right)$ is replaced by an orthogonal matrix Q_i.

In a problem of order $n = pm$, the size of the active set of columns is $p(m-k)$ whereas the size of the inactive set of columns is pk. The algorithm runs on p processors in parallel. Thus, $p = n/m$ is large if m is chosen small. However, the number pk of inactive columns must be smaller than n in order that there is work to do at all. So, one has the constraint

$$m > k. \tag{7.7}$$

Thus, for this algorithm the maximal number of processors which can be exploited occurs for $m = k + 1$, i.e., the maximal amount of parallelism is

$$p_{\max} = \frac{n}{k+1}. \tag{7.8}$$

If the parallelism is maximal, there are only $p = p_{\max}$ columns in the active set, one for each processor. For bidiagonal systems we have $p_{max} = n/2$ which corresponds to the cyclic reduction algorithm of section 6. One could generalize the definition of cyclic reduction to say that it is the method which exploits the maximal amount of parallelism at each step. Note, however, that for tridiagonal systems $p_{\max} = n/3$. This is less than $p_{\max} = n/2$ which is obtained for Cholesky factorization but this is the price to be paid for the increased stability by pivoting.

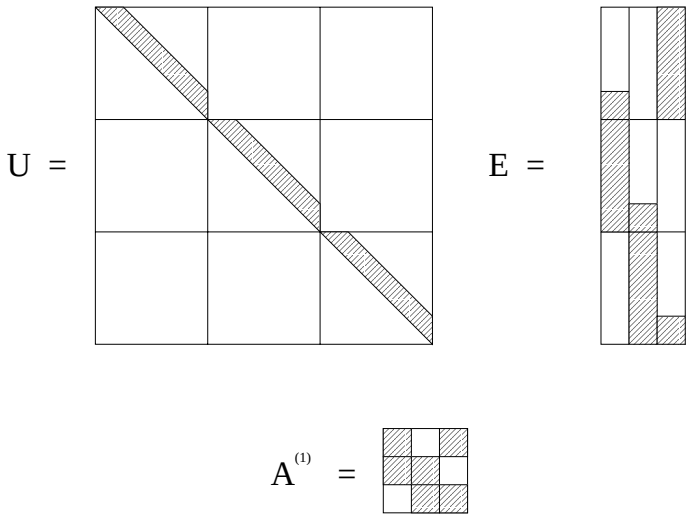

FIGURE 7.3. Matrices involved in a block elimination step

In each step of Gaussian elimination up to k nonzero off-diagonal elements of A are zeroed. This leads to up to k nonzero off-diagonal elements in each column of L_i and thus in L. While these nonzero elements are located in consecutive rows when they are generated, permutations required in later elimination steps can totally destroy this pattern and nothing can be said in general about the position of the k nonzero elements of any column of L_i. A similar remark holds for F_i.

In the case of QR factorization the factors Q_i will in general have a dense upper triangular part and a nonzero band of width k below the diagonal. Usually L_i or Q_i, respectively, are stored in factored form. The nonzero structure of L_i is stored implicitly in the permutations [1].

The nonzero structure of the other factors is more "predictable". After permutation with S and Gaussian elimination with partial pivoting one obtains a factorization of the form given in (4.2). The "U-factor" is of the form

$$\begin{bmatrix} U & E \\ & A^{(1)} \end{bmatrix} \tag{7.9}$$

An upper bound for the nonzero structure of U, E and $A^{(1)}$ is given in Fig. 7.3 for the case $p = 3$. In particular, $A^{(1)}$ is periodic block bidiagonal with k by k blocks. Thus, after the first block elimination step of the banded solver one has to use a block bidiagonal solver for the remaining blocks elimination steps. Note that the upper bounds for the nonzero structure is just the nonzero structure of the matrices which appear in the QR factorization.

Because the amount of partial pivoting in not known a priori, only bounds on the number of floating point operations are obtainable in general in advance of the actual numerical computations. In the unlikely event where no pivoting is required the factorization of the blocks A_{ii} may not need any floating point operations at all. In the worst case the frontal matrix is always (up to the few last elimination steps) of size k by k. Therefore at most $m(2k-1)(k-1)$ floating point operations are needed for the factorization of A_{ii}. An upper bound for the operations required to compute the off-diagonal blocks of E is again $m(2k-1)(k-1)$. Thus, an upper

bound for the parallel complexity of this step is given by

$$t_{par} \leq 2m(2k-1)(k-1)t_F. \tag{7.10}$$

Notice that there is no redundancy introduced by solving the periodically banded linear system in parallel as the last k columns fill up in the serial algorithm as well. As there is no communication required the speedup of this part can be seen to be p. The parallel bottleneck is in the second and later steps which will be discussed in the next section. However, for large enough matrices the main portion of the floating point operations is done in the first step.

The memory requirements of the serial as well as the parallel algorithm are $2kn$ memory locations.

Remark. In this section we presented a parallel band solver suitable for periodically banded matrices. Ordinary banded matrices, i.e. matrices without nonzeros elements in the upper-right and lower-left corner, are forced into this form. It may appear that unnecessary work is introduced in this way. This is however not the case as pivoting is chasing the nonzero elements down the separator again. A slightly different algorithm is obtained, if it is taken into account that the separator doesn't fill up. Essentially the right-most separator is omitted [4]. So, the last processor has less operations to perform. As the work of the other processors stays the same, there is no gain in the parallel execution time. In the sequential algorithm however, the work is halved. Therefore, the double width separator algorithm has redundancy 2 if applied to ordinary banded matrices.

8. The parallel block bidiagonal solver

During the elimination step discussed in the previous section, the block-bidiagonal matrix

$$A^{(1)} = \begin{bmatrix} A^{(1)}_{1,1} & & & A^{(1)}_{1,p} \\ A^{(1)}_{2,1} & \ddots & & \\ & \ddots & \ddots & \\ & & A^{(1)}_{p,p-1} & A^{(1)}_{p,p} \end{bmatrix} \tag{8.1}$$

was produced (see Figure 7.3). The square blocks $A^{(1)}_{i,j}$ all have order k, where k is the bandwidth of A. This first block elimination step did not involve any communication between processors. In the remaining steps, however, communication is required in order to further reduce the matrix.

The parallel block bidiagonal elimination to solve the reduced system $A^{(1)}\mathbf{x}^{(1)} = \mathbf{y}^{(1)}$ proceeds similarly as the scalar algorithm described in section 6. We discuss an implementation of the blocked algorithm that is scalable with respect to memory requirements. For simplicity of presentation we assume that $p = 2^q$.

After the first block elimination step, the i-th block row of $A^{(1)}$ and the corresponding section of the right hand side $\mathbf{y}^{(1)}$ reside in the memory of processor i. To start with the parallel block bidiagonal elimination processors $i-1$ and i, i even, exchange their data. Both, processor $i-1$ and i, form the matrix

$$\begin{bmatrix} A^{(1)}_{i-1,i-1} & 0 & A^{(1)}_{i-1,i-2} & \mathbf{y}^{(1)}_{i-1} \\ A^{(1)}_{i,i-1} & A^{(1)}_{i,i} & 0 & \mathbf{y}^{(1)}_{i} \end{bmatrix}, \qquad i = 2, 4, \ldots, p, \tag{8.2}$$

and perform k steps of Gaussian elimination with partial pivoting[1],

$$P_{i-1}^{(1)} \begin{bmatrix} A_{i-1,i-1}^{(1)} & 0 & A_{i-1,i-2}^{(1)} & \mathbf{y}_{i-1}^{(1)} \\ A_{i,i-1}^{(1)} & A_{i,i}^{(1)} & 0 & \mathbf{y}_{i}^{(1)} \end{bmatrix} = \begin{bmatrix} L_{i-1}^{(1)} & 0 \\ F_{i-1}^{(1)} & I_k \end{bmatrix} \begin{bmatrix} U_{i-1}^{(1)} & D_{i-1}^{(1)} & E_{i-1}^{(1)} & \mathbf{z}_{i-1}^{(1)} \\ 0 & A_{i/2,i/2}^{(2)} & A_{i/2,i/2-1}^{(2)} & \mathbf{y}_{i/2}^{(2)} \end{bmatrix}. \tag{8.3}$$

We proceed similarly if there are several right hand sides. In the sequel, processors $i-1$ and i play different roles. Processor i continues to participate at the forward elimination with the matrices $A_{i/2,i/2}^{(2)}$ and $A_{i/2,i/2-1}^{(2)}$ and the updated right hand side $\mathbf{y}_{i/2}^{(2)}$. Processor i–1 waits until it receives the two sections $\mathbf{x}_i$ and $\mathbf{x}_{i-2}$ of the solution vector from processor i which enables it to calculate $\mathbf{x}_{i+1}$. Here, we tacitly identify $\mathbf{x}_0$ with $\mathbf{x}_p$.

This procedure is executed recursively until, after $q-1 = \log_2(p) - 1$ steps, only the two processors $p/2$ and p are left to solve the $2k \times 2k$ system

$$\begin{bmatrix} A_{1,1}^{(q)} & A_{1,2}^{(q)} \\ A_{2,1}^{(q)} & A_{2,2}^{(q)} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{p/2} \\ \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1^{(q)} \\ \mathbf{y}_2^{(q)} \end{bmatrix} \tag{8.4}$$

by Gaussian elimination with partial pivoting. We write the factorization of the matrix in (8.4) as

$$P_1^{(q)} \begin{bmatrix} A_{1,1}^{(q)} & A_{1,2}^{(q)} \\ A_{2,1}^{(q)} & A_{2,2}^{(q)} \end{bmatrix} = L_1^{(q)} U_1^{(q)}. \tag{8.5}$$

The complete block cyclic reduction is given in Algorithm 1 on page 20. The last loop in this algorithm is necessary to guarantee that all processors have the information needed to continue with the local backward substitution. Although there are $p/2^{l-1}$ processors working on level l, the degree of parallelism is only $p/2^l$ as always two processors work redundantly. Notice that all information of the factorization is still available to solve the block-bidiagonal system with further right hand sides.

After the parallel solution of the periodically block-bidiagonal system, processor i has $\mathbf{x}_i$ and $\mathbf{x}_{i-1}$ stored in its memory. These are the components corresponding to the E-blocks in the i-th block row of the overall system which are precisely the information needed to perform the backward substitution with the matrix U in (7.9). There is no communication during this backward substitution.

The number of floating point operations required by Algorithm 1 depends on the pivoting. Again, we bound this number from above. The (parallel) complexity of each of the l-th elimination step, $l < q$, is at most $3k^2/2 + O(k)$ divisions, $23k^3/6 + O(k^2)$ multiplications, and $23k^3/6 + O(k^2)$ additions. On the top level there are at most $2k^2 + O(k)$ divisions, $8k^3/3 + O(k^2)$ multiplications, and $8k^3/3 + O(k^2)$ additions needed. Backward substitution requires $4k^2 - k$ additions, as many multiplications, and $2k$ divisions on the top level. On the lower levels the number is $k(k-1)/2$ additions and multiplications, respectively, and k divisions. Thus, the

[1]For simplicity of presentation we assume here and in the sequel that indices smaller than 1 and larger than p are mapped onto $[1, \ldots, p]$ by adding or subtracting an appropriate multiple of p. In (8.2), e.g., $A_{i-1,i-2}^{(1)}$ for $i=2$ means $A_{1,p}^{(1)}$.

Algorithm 1 (Parallel block-bidiagonal cyclic reduction). *This algorithm performs forward elimination and backward substitution with one right hand side.*
Input: On processor i, $1 \le i \le p$, matrices $A^{(1)}_{i,i}$, $A^{(1)}_{i,(i-2) \bmod p+1}$ and right hand side $\mathbf{y}^{(1)}_i$.
Output: On processor i, $i \neq p/2, p$, $\mathbf{x}_i$, $\mathbf{x}_{(i-2) \bmod p+1}$ as well as $P^{(r+1)}_{i/2^r}$, $L^{(r+1)}_{i/2^r}$, $F^{(r+1)}_{i/2^r}$, $U^{(r+1)}_{i/2^r}$, $D^{(r+1)}_{i/2^r}$ and $E^{(r+1)}_{i/2^r}$. Here, $r \ge 0$ is the largest integer such that 2^r divides i.
On processor $i = p/2, p$, $\mathbf{x}_i$, $\mathbf{x}_{i-1}$ and $P^{(q)}_1$, $L^{(q)}_1$, $U^{(q)}_1$.

for l **from** 1 **to** $q-1$ **do**
  for i **from** 2^l **by** 2^l **to** p **do**
    processors $i-2^{l-1}$ *and* i *exchange their data and form matrix*

$$\begin{bmatrix} A^{(l)}_{g-1,g-1} & 0 & A^{(l)}_{g,(g-3) \bmod s+1} & \mathbf{y}^{(l)}_{g-1} \\ A^{(l)}_{g,g-1} & A^{(l)}_{g,g} & 0 & \mathbf{y}^{(l)}_{g} \end{bmatrix}, \qquad g := i/2^{l-1}, \quad s := p/2^{l-1}.$$

    processors i *and* $i+2^{l-1}$ *both factor this matrix according to* (8.3)

$$P^{(l)}_{g-1} \begin{bmatrix} A^{(l)}_{g-1,g-1} & 0 & A^{(l)}_{g,(g-3) \bmod s+1} & \mathbf{y}^{(l)}_{g-1} \\ A^{(l)}_{g,g-1} & A^{(l)}_{g,g} & 0 & \mathbf{y}^{(l)}_{g} \end{bmatrix} = \begin{bmatrix} L^{(l)}_g & 0 \\ F^{(l)}_g & I \end{bmatrix} \begin{bmatrix} U^{(l)}_g & D^{(l)}_g & E^{(l)}_g & \mathbf{z}^{(l)}_g \\ 0 & A^{(l+1)}_{g/2,g/2} & A^{(l+1)}_{g/2,(g/2-3) \bmod (s/2)+1} & \mathbf{y}^{(l+1)}_{g/2} \end{bmatrix}.$$

  endfor
endfor
processors $p/2$ *and* p *exchange their data, form*

$$\begin{bmatrix} A^{(q)}_{1,1} & A^{(q)}_{1,2} & \mathbf{y}^{(q)}_1 \\ A^{(q)}_{2,1} & A^{(q)}_{2,2} & \mathbf{y}^{(q)}_2 \end{bmatrix}$$

and solve the system (8.4) *according to* (8.5)
for l **from** $q-1$ **by** -1 **to** 1 **do**
  for i **from** 2^{l-1} **by** 2^l **to** $p - 2^{l-1}$ **do**
    processor i *receives* $\mathbf{x}_{i-2^{l-1}}$ *and* $\mathbf{x}_{i+2^{l-1}}$ *from processor* $i-2^{l-1}$
    and computes $\mathbf{x}_i$ *satisfying*

$$U^{(l)}_{g-1}\mathbf{x}_i = \mathbf{z}^{(l)}_g - D^{(l)}_g \mathbf{x}_{i-2^{l-1}} - E^{(l)}_g \mathbf{x}_{i+2^{l-1}}, \qquad g := i/2^{l-1}.$$

  endfor
endfor
for i **from** 2 **by** 2 **to** p **do**
  processor i *receives* $\mathbf{x}_{i-1}$ *from processor* $i-1$.
endfor

parallel complexity of the block CR algorithm is approximately

$$t = \frac{23}{3} \log_2(p) k^3 t_F, \tag{8.6}$$

if communication is not taken into account. This bound slightly overestimates the execution time as the top level only requires around $16k^3/3$ operations instead of

$23/3k^3$ but for the purposes of analysis of parallel performance this bound is good enough. On one processor the execution time of the algorithm is

$$t \approx \frac{23}{3} p k^3 t_F. \tag{8.7}$$

As there is no redundant work done in the parallel implementation, the parallel speedup for solving a system of order $p(k-1)$ becomes

$$S = p/\log_2(p). \tag{8.8}$$

So, with 1024 processors a speedup of $S \approx 102$ could be achieved. Here, we have compared CR with the serial algorithm to solve the block-bidiagonal system.

Combining (7.10) and (8.6), the solution of the overall banded linear system on p processors has a time complexity

$$t(p) = \Big(2\frac{n}{p}(2k-1)(k-1) + \frac{23}{3}\log_2(p)k^3 + O(k^3)\Big)t_F. \tag{8.9}$$

Omitting lower order terms, we obtain a speedup

$$S(p) = \frac{t(1)}{t(p)} \approx \frac{p}{1 + \frac{23k}{12n}\, p \log_2(p)}. \tag{8.10}$$

Note that the speedup does not increase monotonically with p, as the denominator in (8.10) grows faster than the numerator. In fact, $S(p)$ grows until $p_{\text{opt}} = (12\log(2)/23)(n/k) \approx n/(3k)$ at which point the highest possible speedup is

$$S_{\text{opt}} = S(p_{\text{opt}}) \approx \frac{n}{3k} \cdot \frac{1}{1 + \log_2\left(\frac{n}{3k}\right)}.$$

In our actual implementation, we store the block rows including the respective right hand(s) side as indicated in (8.2) in a work array. When sending data in the forward elimination phase we simultaneously send and receive complete block rows of the work array. These are single long messages. The disadvantage of this proceeding is that k^2 zero values are sent. This is justified as the communication startup cost is the dominant factor when sending short messages. If the work array is stored properly, the message does not even have to be composed in a message buffer. In the backward substitution phase the messages have length $2k$. As before, we model the communication time for transferring a message of length n data items by $\sigma + \tau n = (\mu + \nu n)t_F$. We do not neglect τ (or ν) here, as the message volume $3k^2$ can be considerable. In the back-substitution phase the message lengths are negligible. With communication the execution time of the band solver becomes

$$t(p) \approx \Big(2\frac{n}{p}(2k-1)(k-1) + \big(\frac{23}{3}k^3 + 2\mu + 3k^2\nu\big)\log_2(p)\Big)t_F. \tag{8.11}$$

This implies the speedup

$$S(p) \approx \frac{p}{1 + \left(\frac{23k}{12n} + \frac{\mu}{2nk^2} + \frac{3\nu}{4n}\right)\, p \log_2(p)}. \tag{8.12}$$

The importance of the communication startup time relative to the communication bandwidth becomes small if k increases. In order to get respectable speedups the matrix order must be large compared to the bandwidth. If $n = 100000$, $k = 50$, $\mu = 1000$, $\nu = 1$ the speedup for 128 processors is 68.6 with communication taken into account and 68.9 without.

9. Performance

The algorithm presented in sections 7 and 8 has been implemented using Fortran 77 and MPI. It has been tested on the Fujitsu AP1000 with 128 processors at the ANU in Canberra, on the Intel Paragon with 150 processors at the ETH in Zurich, and on the SGI Power Challenge with 14 processors in Canberra. The AP1000 and the Paragon are distributed memory machines, while the Power Challenge has a shared memory.

Three different problem sizes were considered:

1. A small problem with dimension $n = 20000$ and upper and lower bandwidth 10, such that $k = 20$.
2. An intermediate problem with dimension $n = 100,000$ and the same bandwidth as in 1., i.e., $k = 20$.
3. A large problem with dimension $n = 100,000$ and upper and lower bandwidth 50, i.e., $k = 100$.

The matrices generated were diagonally dominant Toeplitz matrices. Note that the algorithm cannot exploit diagonal dominance. In fact, we see from the structure of the permuted matrix AS, cf. Fig. 7.2, that partial column pivoting forces swapping rows in every elimination step. As the largest element in a column is $(k-1)/2$ elements away from the diagonal the bandwidth of U will be $3k/2$. We therefore believe that the algorithm performs about as in the average. While the algorithm can deal with periodically banded matrices only "ordinary" banded matrices where used in the tests. As the work on a single processor is now reduced by a factor two, also the speedup in (8.12) is reduced by that factor.

On one processor the LAPACK routines "DGBTF2" and "DGBTRS" were used to factorize and solve the banded linear system [1]. It is assumed that these LAPACK routines are very efficient on one node and in all cases they are using specially optimized BLAS routines.

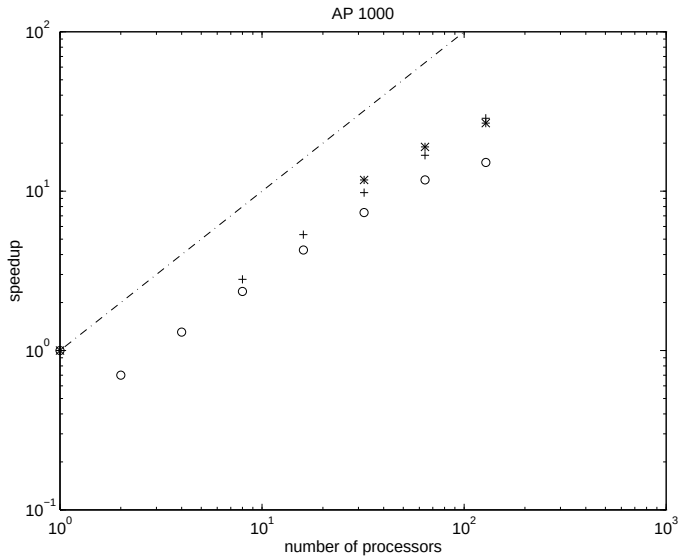

Figure 9.1. Speedup of parallel solvers compared to LAPACK routines. o: small problem, +: intermediate problem and *: large problem. The dash-dotted line indicates ideal speedup.

In Figure 9.1 the speedups on the AP1000 are displayed. The AP1000 has 128 SPARC processors. On two processors the parallel code takes longer than the serial code due to redundancy. After that the speedup is fairly linear but deteriorates, especially for small problem sizes. This is clear as in small problems the first block elimination step is relatively cheap compared with the block cyclic reduction of the reduced system the order of which depends linearly on p but not on n. Clearly, speedup increases with n if p and k is fixed. Looking at the two larger problems, the speedup is larger for the wider banded problem for $p \leq 64$. This indicates a large influence of μ in (8.12), i.e., that the communication latency is dominated by the startup time. ν is small. So, the interprocessor communication bandwidth is high compared with the performance of the CPU's floating point unit. For the intermediate and the large problem size the times on one processor were extrapolated from the one-processor performance of a smaller problem with equal bandwidth k using the assumption that the time is proportional to nk^2. These problems are too large to be solved on one processor due to memory size limitations.

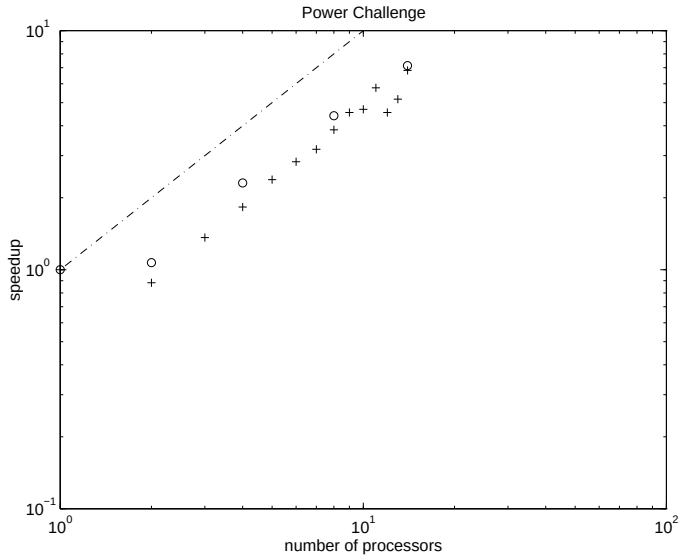

FIGURE 9.2. Speedup of parallel solvers compared to LAPACK routines. o: small problem, +: intermediate problem. The dash-dotted line indicates ideal speedup.

In Figure 9.2 the speedup on the Power Challenge of Silicon Graphics is displayed. Only 14 (non-dedicated) processors were available, but in that range both the small and the intermediate problem showed speedups linear in the number of processors. The efficiency $S(p)/p$ is close to constant in the processor range depicted. This indicates that the communication latency is small. Evidently, MPI is implemented such that it can make use of the shared memory. The intermediate sized problem was too big to fit in the memory of one processor. So, the time to solve the intermediate problem on one processor was extrapolated from a smaller problem. The largest problem could not even be computed on 14 processors. On the Power Challenge the effect of the redundancy appears to be lower than on the AP1000. We explain this by the fact that the observed MFlop/s rates are higher for the forward and backward substitution than for the LU factorization. The redundant floating point operations for computing the matrix E in (7.9) are spent in forward and backward substitution.

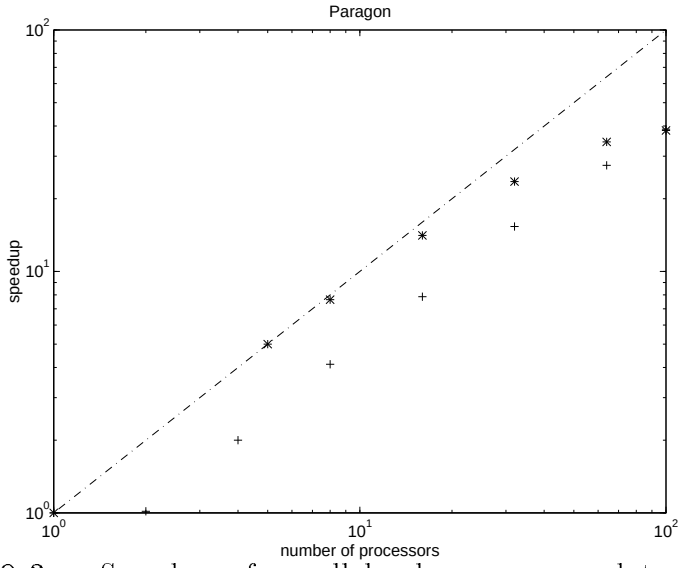

FIGURE 9.3. Speedup of parallel solvers compared to LAPACK routines. +: intermediate problem, *: large problem. The dash-dotted line indicates ideal speedup.

In Figure 9.3 the speedup on the Intel Paragon using up to 100 processors is displayed. Only the intermediate and the large problem sizes were considered. While the speedup for the larger problem size is substantially higher for small processor numbers, it does deteriorate more rapidly from the curve of ideal speedup as p increases. This is due to much larger reduced system. Again the expensive communication startup is noticeable by the higher speedups of the larger problems with smaller processor numbers. The one-processor execution time has again be estimated. As with the Power Challenge, the redundancy is less than what might be expected.

10. Conclusions

We have shown that distributed memory parallel band solvers implementing Gaussian elimination with partial pivoting are feasible and produce speedups which are comparable to the ones obtained from LU without pivoting or Cholesky factorization [4, 3]. Of course the execution times are shorter for the latter algorithms. Speedup is affected first of all by redundancy introduced in the parallelization of the algorithm. If the processor number grows while the problem size remains fixed speedup deteriorates due to increasing communication costs. However, the complexity analysis indicates that if the problem size is increased like $p\log(p)$, the efficiency is independent on the processor number p.

Portable software has been developed using message passing and tests were run on the Intel Paragon, Silicon Graphics Power Challenge and Fujitsu AP1000. The tests confirm the analysis and show good scalability.

Acknowledgements

Much of the work was done while the second author was visiting the Institute of Scientific Computing in Zürich. The research of the second author was partially

supported by the Swiss Federal Institute of Technology and the Advanced Computational Systems CRC, Australia.

References

1. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson, *LAPACK users' guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.
2. I. J. Anderson and S. K. Harbour, *Parallel factorization of banded linear matrices using a systolic array processor*, Advances in Computational Mathematics **5** (1996), no. 1, 1–14.
3. P. Arbenz, *On experiments with a parallel direct solver for diagonally dominant banded linear systems*, Euro-Par '96 (L. Bougé, P. Fraigniaud, A. Mignotte, and Y. Robert, eds.), Springer-Verlag, Berlin, 1996, (Lecture Notes in Computer Science, 1124), pp. 11–21.
4. P. Arbenz and W. Gander, *A survey of direct parallel algorithms for banded linear systems*, Tech. Report 221, Departement Informatik, Institut für Wissenschaftliches Rechnen, ETH Zürich, 1994.
5. M. Berry and A. Sameh, *Multiprocessor schemes for solving block tridiagonal linear systems*, The INterantional Journal of Supercomputer Applications **2** (1988), no. 3, 37–57.
6. R. Brent, A. Cleary, M. Dow, M. Hegland, J. Jenkinson, Z. Leyk, M. Nakanishi, M. Osborne, P. Price, S. Roberts, and D.Singleton, *Implementation and performance of scalable scientific library subroutines on Fujitsu's VPP500 parallel-vector supercomputer*, Proceedings of the Scalable High-Performance Computing Conference, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 526–533.
7. O. Buneman, *A compact non-iterative Poisson solver*, Tech. Report 294, Institute for Plasma Research, Stanford University, Stanford, CA, 1969.
8. B. L. Buzbee, G. H. Golub, and C. W. Nielson, *On direct methods for solving Poisson's equation*, SIAM Journal on Numerical Analysis **7** (1970), no. 4, 627–656.
9. J. M. Conroy, *Parallel algorithms for the solution of narrow banded systems*, Applied Numerical Mathematics **5** (1989), no. 5, 409–421.
10. J. J. Dongarra and L. Johnsson, *Solving banded systems on a parallel processor*, Parallel Computing **5** (1987), no. 1-2, 219–246.
11. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices*, Clarendon Press, Oxford, 1986.
12. C. Dun, M. Hegland, and M. Osborne, *Parallel stable solution methods for tridiagonal linear systems of equations*, Computational Techniques and Applications: CTAC-95 (R. L. May and A. K. Easton, eds.), World Scientific, Singapore, 1996, pp. 267–274.
13. G. H. Golub and C. F. Van Loan, *Matrix computations*, 2nd ed., The Johns Hopkins University Press, Baltimore, MD, 1989.
14. A. Y. Grama, A. Gupta, and V. Kumar, *Isoefficiency: Measuring the scalability of parallel algorithms and architecture*, IEEE Parallel & Distributed Technology: Systems & Applications **1** (1993), no. 3, 12–21.
15. A. Gupta, F. G. Gustavson, M. Joshi, and S. Toledo, *The design, implementation, and evaluation of a banded linear solver for distributed-memory parallel computers*, Research Report RC 20481, IBM T. J. Watson Research Center, Yorktown Heights, NY, June 1996.
16. I. N. Hajj and S. Skelboe, *A multilevel parallel solver for block tridiagonal and banded linear systems*, Parallel Computing **15** (1990), no. 1-3, 21–45.
17. M. Hegland, *On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorization*, Numerische Mathematik **59** (1991), no. 5, 453–472.
18. R.W. Hockney, *A fast direct solution of Poisson's equation using Fourier analysis*, Journal of the Association for Computing Machinery **12** (1965), no. 1, 95–113.
19. K. Hwang, *Advanced computer architecture: Parallelism, scalability, programmability*, McGraw-Hill, New York, NY, 1993.
20. S. L. Johnsson, *Solving narrow banded systems on ensemble architectures*, ACM Transactions on Mathematical Software **11** (1985), no. 3, 271–288.
21. J. Larriba-Pey, J. Navarro, A. Jorba, and O. Roig, *Review of general and Toeplitz bidiagonal solvers*, Parallel Computing **to appear** (1996).

22. D. Lawrie and A. Sameh, *The computation and communication complexity of a parallel banded system solver*, ACM Transactions on Mathematical Software **10** (1984), no. 2, 185–195.
23. C. L. Lawson and R. J. Hanson, *Solving least squares problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
24. U. Meier, *A parallel partition method for solving banded systems of linear equations*, Parallel Computing **2** (1985), no. 1, 33–43.
25. T.-H. Oleson and J. R. Gilbert, *One-way dissection with pivoting on the hypercube*, In: Parallel Sparse Matrix Algorithms by T.-H. Oleson, Tech. Report No. 76, Department of Informatics, University of Bergen, Bergen, Norway, December 1992, 38 pages.
26. J. M. Ortega, *Introduction to parallel and vector solution of linear systems*, Plenum Press, New York, NY, 1988.
27. M. R. Osborne, *On shooting methods for boundary value problems*, Journal of Mathematical Analysis and Applications **27** (1969), 417–433.
28. Y. Saad and M. Schultz, *Parallel direct methods for solving banded linear systems*, Linear Algebra and its Applications **88/89** (1987), 623–650.
29. A. Sameh and D. Kuck, *On stable parallel linear system solvers*, Journal of the Association for Computing Machinery **25** (1978), no. 1, 81–91.
30. A. H. Sameh and D. J. Kuck, *A parallel QR algorithm for symmetric tridiagonal matrices*, IEEE Transactions on Computers **C-26** (1977), no. 2, 147–151.
31. J. H. Wilkinson, *The algebraic eigenvalue problem*, Clarendon Press, Oxford, 1965.
32. S. J. Wright, *Parallel algorithms for banded linear systems*, SIAM Journal on Scientific and Statistical Computing **12** (1991), no. 4, 824–842.
33. ———, *Stable parallel algorithms for two-point boundary value problems*, SIAM Journal on Scientific and Statistical Computing **13** (1992), no. 3, 742–764.
34. ———, *A collection of problems for which Gaussian elimination with partial pivoting is unstable*, SIAM Journal on Scientific Computing **14** (1993), no. 1, 231–238.

Swiss Federal Institute of Technology (ETH), Institute of Scientific Computing, 8092 Zurich, Switzerland
E-mail address: arbenz@inf.ethz.ch

Computer Sci. Lab., RSISE, Australian National University, Canberra ACT 0200, Australia
E-mail address: Markus.Hegland@anu.edu.au