The stable parallel solution of narrow banded linear systems

Peter Arbenz* Markus Hegland†

Abstract

We report on experiments with a stable algorithm for the parallel solution of banded and periodically banded linear systems. In contrast to most of the known parallel algorithms which are stable only for symmetric positive definite or diagonally dominant systems, the new algorithm incorporates pivoting without sacrificing efficiency. We report on numerical experiments conducted on an Intel Paragon, a Silicon Graphics Power Challenge and a Hewlett Packard Exemplar.

1 Introduction

Methods will be discussed for the solution of linear systems

$$Ax = b$$

where the $n \times n$ system matrix A is banded with upper bandwidth k_u and lower bandwidth k_l, i.e., $\alpha_{ij} \neq 0$ implies $-k_l \leq i-j \leq k_u$. 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 $j \leq i-n+k_u$. We call these matrices *periodically banded.* For example, the nonzero structure of a periodically tridiagonal matrix is

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

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 [12, 11, 13, 7, 9, 15]. In these cases Gaussian elimination is stable without pivoting. If pivoting has been included efficiency was lost [6, 17]. An alternative to Gaussian elimination with pivoting is QR factorization [2, 16]. Further articles can be found on the closely related block tridiagonal systems [5].

*Swiss Federal Institute of Technology (ETH), Institute of Scientific Computing, 8092 Zurich, Switzerland

†Australian National University, Computer Sci. Lab., RSISE, Canberra ACT 0200, Australia. The research of this author was partially supported by the Swiss Federal Institute of Technology and the Advanced Computational Systems CRC, Australia.

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 [8], this situation is considered for the cases where no pivoting is needed.

2 The algorithm

The parallel band solver discussed in the sequel has been presented first in [10] and has been thoroughly investigated in [4]. The algorithm has similarities with the one-way dissection algorithm in [14] for sparse linear systems. It consists of a succession of steps which can be interpreted as LU factorizations. While the second and later steps all factorize periodic 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 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 (and, by consequence, every banded) matrix a circular shift S_0 exists such that $S_0 A$ (or, equivalently, AS_0) is in standard form. Note that the standard form is not uniquely defined. In the following we assume that A is in *standard form* with lower bandwidth k.

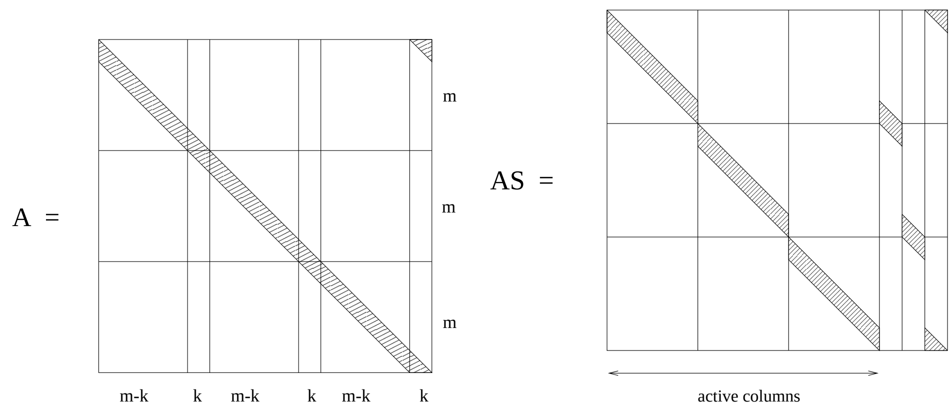

FIG. 1. *Partitioning of original banded matrix A and of permuted matrix AS*

For simplicity of presentation we assume that $n = pm$. Then, the set of rows of A can be partitioned into p equally sized sets. The columns in turn are partitioned in blocks of size $(m-k, k, m-k, k, m-k, \ldots)$. In the case $p = 3$ this leads to a blocking of A as displayed in Fig. 1. In general the partitioned matrix has the following basic structure,

$$A = \begin{bmatrix} A_{11} & A_{12} & & & & & & & A_{1p} \\ & A_{21} & A_{22} & A_{23} & & & & & \\ & & & A_{32} & A_{33} & A_{34} & & & \\ & & & & & & \ddots & & \\ & & & & & & A_{p,p-1} & A_{pp} & A_{p,p+1} \end{bmatrix}, \qquad \begin{array}{r} A_{i,i-1} \in \mathbb{R}^{m\times k}, \\ A_{i,i+1} \in \mathbb{R}^{m\times k}, \\ A_{i,i} \in \mathbb{R}^{m\times m-k}. \end{array}$$

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 [11].

The block elimination scheme will eliminate the unknowns corresponding to the columns in the $m \times m - k$ blocks. These *active columns* have nonzero elements only in one partition, i.e., on one processor. Furthermore, due to the row-based partitioning, all the rows corresponding to these nonzero elements reside on the same processor. Therefore, this first block elimination step consists of p independent block eliminations requiring no interprocessor communication. Let the factorization

$$(1)\quad P_iA_{ii} = \begin{bmatrix} L_i & 0 \\ F_i & I_{k\times k} \end{bmatrix} \begin{bmatrix} U_i \\ 0 \end{bmatrix}, \quad U_i, L_i \in \mathbb{R}^{m-k\times m-k}, F_i \in \mathbb{R}^{k\times m-k}, \quad i = 1,\ldots,p,$$

be obtained by Gaussian elimination with partial pivoting. The off-diagonal blocks are then multiplied with P_i and the inverse of the "L-factor" of A_{ii} to get

$$(2)\qquad \begin{pmatrix} E_{i,i-1} \\ A^{(1)}_{i,i-1} \end{pmatrix} := \begin{pmatrix} L_i & \\ F_i & I \end{pmatrix}^{-1} P_iA_{i,i-1} = \begin{pmatrix} L_i^{-1} & \\ -F_iL_i^{-1} & I \end{pmatrix} P_iA_{i,i-1},$$

$$(3)\qquad \begin{pmatrix} E_{i,i} \\ A^{(1)}_{i,i} \end{pmatrix} := \begin{pmatrix} L_i & \\ F_i & I \end{pmatrix}^{-1} P_iA_{i,i+1} = \begin{pmatrix} L_i^{-1} & \\ -F_iL_i^{-1} & I \end{pmatrix} P_iA_{i,i+1}.$$

Let S be the matrix that permutes the separator columns of A to the right, cf. Fig. 1. Define $P = \oplus_{i=1}^{p} P_i$, $L = \oplus_{i=1}^{p} L_i$, $F = \oplus_{i=1}^{p} F_i$, $U = \oplus_{i=1}^{p} U_i$,

$$(4)\qquad E = \begin{bmatrix} E_{11} & & & E_{1p} \\ E_{21} & E_{22} & & \\ & \ddots & \ddots & \\ & & E_{p,p-1} & E_{pp} \end{bmatrix}, \qquad E_{ij} \in \mathbb{R}^{m\times k},$$

and

$$(5)\qquad A^{(1)} = \begin{bmatrix} A^{(1)}_{11} & & & A^{(1)}_{1p} \\ A^{(1)}_{21} & A^{(1)}_{22} & & \\ & \ddots & \ddots & \\ & & A^{(1)}_{p,p-1} & A^{(1)}_{pp} \end{bmatrix}, \qquad A^{(1)}_{ij} \in \mathbb{R}^{k\times k}.$$

Then, equations (1), (2) and (3) yield the overall factorization

$$(6)\qquad PAS = \begin{bmatrix} L & \\ F & I \end{bmatrix} \begin{bmatrix} U & E \\ & A^{(1)} \end{bmatrix}.$$

An "upper bound" for the nonzero structure of U, E and $A^{(1)}$ is given in Fig. 2 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.

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 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 either the factorization of A_{ii} or the computation of the off-diagonal blocks of E. Thus, an upper bound for the parallel complexity of this step is

$$t_{1st\ step}(p) \le 2m(2k-1)(k-1) \text{ flop}, \qquad m = n/p.$$

Notice that there is no redundancy introduced when solving a periodically banded linear system in parallel as the last k columns fill up in the serial algorithm as well. However, if

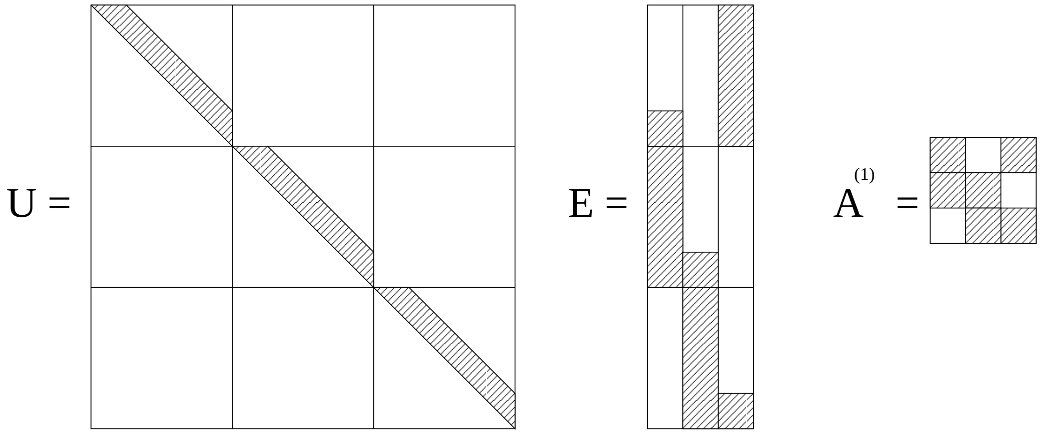

FIG. 2. *Matrices involved in a block elimination step*

A is an ordinary banded matrix, the computation of E is a parallelization overhead! The redundancy in this latter case is around two [3, 4].

Similarly, the memory requirements of the serial as well as the parallel algorithm are $3kn$ memory locations if the original A is periodically banded. In the case of an ordinary banded matrix the serial algorithm requires $(k_u + 2k_l)n$ memory location.

The difficult part of the algorithm are the second and later elimination steps in which the block-bidiagonal reduced system

$$A^{(1)}\mathbf{x}^{(1)} = \mathbf{y}^{(1)} \tag{7}$$

has to be solved. Here, $A^{(1)}$ is the matrix in (5). This part of the algorithm requires communication. 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

$$\begin{bmatrix} A^{(1)}_{i-1,i-1} & 0 & A^{(1)}_{i-1,i-2} & \mathbf{y}^{(1)}_{i-1} \\ A^{(1)}_{i,i-1} & A^{(1)}_{i,i} & 0 & \mathbf{y}^{(1)}_{i} \end{bmatrix}, \qquad i = 2, 4, \ldots, p, \tag{8}$$

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

$$\begin{aligned} P^{(1)}_{i-1} &\begin{bmatrix} A^{(1)}_{i-1,i-1} & 0 & A^{(1)}_{i-1,i-2} & \mathbf{y}^{(1)}_{i-1} \\ A^{(1)}_{i,i-1} & A^{(1)}_{i,i} & 0 & \mathbf{y}^{(1)}_{i} \end{bmatrix} \\ &= \begin{bmatrix} L^{(1)}_{i-1} & 0 \\ F^{(1)}_{i-1} & I_k \end{bmatrix} \begin{bmatrix} U^{(1)}_{i-1} & D^{(1)}_{i-1} & E^{(1)}_{i-1} & \mathbf{z}^{(1)}_{i-1} \\ 0 & A^{(2)}_{i/2,i/2} & A^{(2)}_{i/2,i/2-1} & \mathbf{y}^{(2)}_{i/2} \end{bmatrix}. \end{aligned} \tag{9}$$

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

[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), e.g., $A^{(1)}_{i-1,i-2}$ for $i=2$ means $A^{(1)}_{1,p}$.

the matrices $A^{(2)}_{i/2,i/2}$ and $A^{(2)}_{i/2,i/2-1}$ and the updated right hand side $\mathbf{y}^{(2)}_{i/2}$. 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

$$\begin{bmatrix} A^{(q)}_{1,1} & A^{(q)}_{1,2} \\ A^{(q)}_{2,1} & A^{(q)}_{2,2} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{p/2} \\ \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{y}^{(q)}_1 \\ \mathbf{y}^{(q)}_2 \end{bmatrix} \tag{10}$$

by Gaussian elimination with partial pivoting. As soon as processors $p/2$ and p have solved (10), they send the result to $p/4$ and $3p/4$, respectively, which enables them to compute $\mathbf{x}_i$, $i = p/4, 3p/4$. 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 [4]. 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 (6). There is no communication during this final backward substitution.

The number of floating point operations required by the overall algorithm 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 $23k^3/3 + O(k^2)$ flops. On the top level there are at most $16k^3/3 + O(k^2)$ flops needed. The backward substitution is comparatively negligible.

We model the communication time for transferring a message of length n data items by $\sigma + \tau n = (\mu + \nu n)t_F$ where σ is the communication startup time and τ is the reciprocal of the bandwidth of the communication network. μ and ν are the same numbers relative to the execution time t_F of a floating point operation. In our actual implementation, we store the block rows including the respective right hand(s) side as indicated in (8) 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 $2k$.

The execution time of the complete band solver executed on p processors, thus is

$$t(p) \approx \Big(2\frac{n}{p}(2k-1)(k-1) + (\frac{23}{3}k^3 + 2\mu + 3k^2\nu)\log_2(p)\Big)t_F. \tag{11}$$

This implies the speedup

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

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.

3 Performance

The algorithm presented above has been implemented using Fortran 77 and MPI. It has been tested on various multiprocessor computers. We report on the results we have obtained with the 160 processor Intel Paragon at ETH in Zurich, with the 20 processor SGI Power Challenge at ANU in Canberra and with an 8 processor HP Exemplar SPP 1600 at ETH and a 16 processor HP Exemplar SPP 1600 at Hewlett Packard's benchmark center in Dallas. The Paragon is a distributed memory machines, the other two machines have a shared memory.

Three different problem sizes were considered:

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

The test matrices generated were random matrices. However, for a fixed problem size it remained unaltered with varying processor number. 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 (12) is reduced by this factor. On one processor the LAPACK [1] routines `dgbtf2` and `dgbtrs` were used to factor and solve the banded linear system. The solutions obtained with the parallel algorithm were as accurate as the ones obtained with the standard subroutines on one processor.

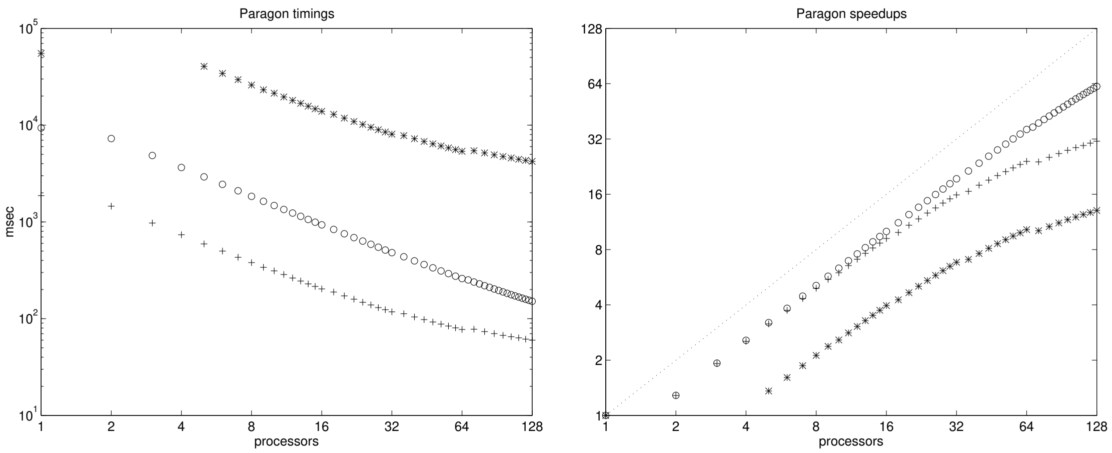

FIG. 3. *Times and speedups of parallel solvers compared to LAPACK routines.* +: *small problem,* o: *intermediate problem,* *: *large problem. The dash-dotted line indicates ideal speedup.*

In Fig. 3 execution times and speedups obtained on the Intel Paragon using up to 128 processors are displayed in logarithmic scale. A Paragon processor performs at a rate of around 20 Mflop/s in the first elimination step of our algorithm. The communication startup time is around 60μs and the network bandwidth is about 130 MB/s. The speedups are as expected from (12). The speedups with bandwidth $k = 10$ are good. They grow almost linearly for the intermediate problem size. The difference to the ideal speedup curve is caused by the high redundancy. The speedups deteriorate with the smallest and the largest problem. In these cases the size n/p of the local problems are small with respect to the bandwidth that appears quadratically in (11) and (12). The one-processor time for the largest problem had to be extrapolated. The problem was too big to fit in the memory of one processor. The drop by a factor 2-3 in speedup is caused by the wider band.

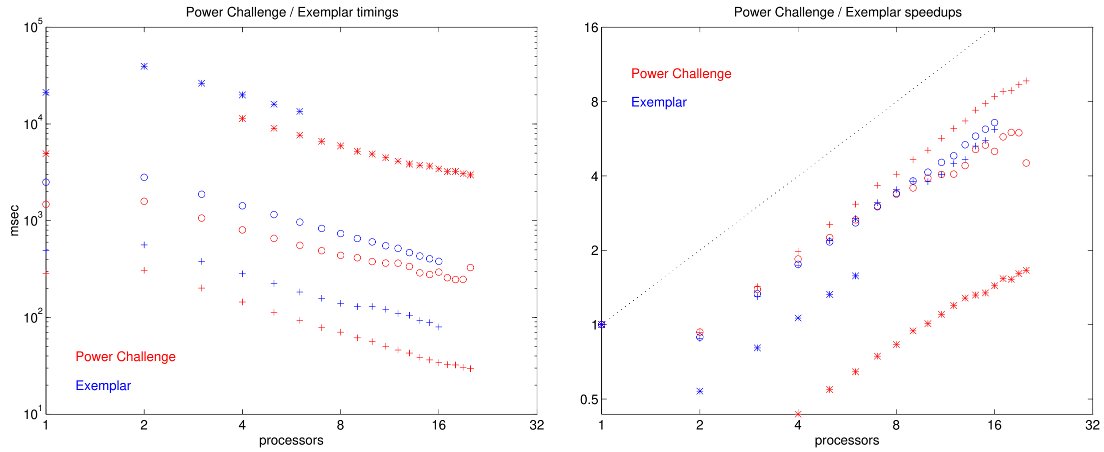

FIG. 4. *Times and speedups of parallel solvers compared to LAPACK routines. +: small problem, o: intermediate problem, *: large problem. The dotted line indicates ideal speedup.*

In Fig. 4 execution times and speedups for the HP Exemplar and the SGI Power Challenge, respectively, are displayed. The large problem was too big to fit in the memory of one processor of the Power Challenge. Therefore, the time to solve the large problem on one processor was linearly extrapolated from a smaller problem with equal bandwidth. The Exemplar was equipped with enough memory to solve all problems. Although the machines have a shared memory the same program as on the Paragon was run. The calls to MPI routines are translated into memory-memory copies by the compiler such that the communication startup time is small. The experiments on the HP were executed in dedicated mode. The SGI was only accessible in non-dedicated mode, but the machine was merely lightly loaded at the time of the measurements. The execution times of the Power Challenge are shorter than those of the Exemplar by factors 1.6 to 2. The Mflop/s rate of the Exemplar and the Power Challenge are around 50 and 100 for the first step of our algorithm. The communication startup times are only a few μsec. The bandwidth that is obtained with MPI is around 80 MB/s on the Exemplar and even less on the Power Challenge.

For $p \geq 2$ the speedups obtained with the Exemplar as well as the Power Challenge are close to linear for the small number of processors considered. This almost linear dependency on p indicates that the weight of the block-bidiagonal solver (including communication) is not yet large. The most striking difference between the shared memory machines and the Paragon is the much larger effect of the redundancy with the former. With a wide band it is quite hard to get a speedup over the serial algorithm at all. This means that the parallelization overhead introduced by the computation of the matrix E in (4) is relatively larger. We assume that the performance difference (in Mflop/s) for LU factorization and forward/backward substitution for computing the portions of E in (2) and (3) is smaller on the SGI and the HP than on the Paragon.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. 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), pp. 1–14.

[3] P. Arbenz and W. Gander, *A survey of direct parallel algorithms for banded linear systems*, Tech. Report 221, ETH Zürich, Computer Science Department, November 1994. `ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/221.ps.gz`.

[4] P. Arbenz and M. Hegland, *The stable parallel solution of general narrow banded linear systems*, Tech. Report 252, ETH Zürich, Computer Science Department, November 1996. `ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/2xx/252.ps.gz`.

[5] M. Berry and A. Sameh, *Multiprocessor schemes for solving block tridiagonal linear systems*, The International Journal of Supercomputer Applications, 2 (1988), pp. 37–57.

[6] J. M. Conroy, *Parallel algorithms for the solution of narrow banded systems*, Applied Numerical Mathematics, 5 (1989), pp. 409–421.

[7] J. J. Dongarra and L. Johnsson, *Solving banded systems on a parallel processor*, Parallel Computing, 5 (1987), pp. 219–246.

[8] 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.

[9] I. N. Hajj and S. Skelboe, *A multilevel parallel solver for block tridiagonal and banded linear systems*, Parallel Computing, 15 (1990), pp. 21–45.

[10] M. Hegland, *Divide and conquer for the solution of banded linear systems of equations*, in Proceedings of the Fourth Euromicro Workshop on Parallel and Distributed Processing, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 394–401.

[11] S. L. Johnsson, *Solving narrow banded systems on ensemble architectures*, ACM Transactions on Mathematical Software, 11 (1985), pp. 271–288.

[12] D. Lawrie and A. Sameh, *The computation and communication complexity of a parallel banded system solver*, ACM Transactions on Mathematical Software, 10 (1984), pp. 185–195.

[13] U. Meier, *A parallel partition method for solving banded systems of linear equations*, Parallel Computing, 2 (1985), pp. 33–43.

[14] T.-H. Oleson and J. R. Gilbert, *One-way dissection with pivoting on the hypercube*, in 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.

[15] J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, NY, 1988.

[16] A. Sameh and D. Kuck, *On stable parallel linear system solvers*, Journal of the Association for Computing Machinery, 25 (1978), pp. 81–91.

[17] S. J. Wright, *Parallel algorithms for banded linear systems*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 824–842.