A survey of direct parallel algorithms for banded linear systems

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Wissenschaftliches Rechnen

Peter Arbenz
Walter Gander

A Survey of Direct Parallel Algorithms for Banded Linear Systems

October 1994

221

ETH Zürich
Departement Informatik
Institut für Wissenschaftliches Rechnen
Prof. Dr. W. Gander

Peter Arbenz
Institut für Wissenschaftliches Rechnen
Eidgenössische Technische Hochschule
CH-8092 Zürich
e-mail: arbenz@inf.ethz.ch

Walter Gander
Institut für Wissenschaftliches Rechnen
Eidgenössische Technische Hochschule
CH-8092 Zürich
e-mail: gander@inf.ethz.ch

This report is also available via anonymous ftp from ftp.inf.ethz.ch
as doc/tech-reports/1994/221.ps.

A Survey of Direct Parallel Algorithms for Banded Linear Systems

Peter Arbenz and Walter Gander

Institut für Wissenschaftliches Rechnen
Eidgenössische Technische Hochschule
CH-8092 Zürich
[arbenz,gander]@inf.ethz.ch

Abstract

We investigate direct algorithms to solve linear banded systems of equations on MIMD multiprocessor computers with distributed memory. We compare the coarse-grain parallel algorithms with ordinary one-processor Gaussian elimination. The parallel algorithms behave satisfactory only if the ratio of bandwidth and matrix order is very small. As a result of the high redundancy of the parallel algorithms efficencies are in general not high. The theoretical considerations are complemented with numerical experiments performed on the Intel Paragon.

Keywords. Banded Linear Systems, Divide and Conquer, Parallel Computation, Scalability.

1 Introduction

In this paper we discuss using direct methods on parallel computers to solve a system of linear equations

$$A\mathbf{x} = \mathbf{b} \tag{1.1}$$

where A is a real banded $n \times n$ matrix with lower half-bandwidth r and upper half-bandwidth s,

$$a_{ij} = 0 \qquad \text{for } i - j > r \text{ and } j - i > s. \tag{1.2}$$

We assume that the matrix A has a *narrow* band, such that $r + s \ll n$. Only in this case is it worth taking into account the zero structure of A, i.e. storing the matrix by diagonals [1] and adapting programs to this storing scheme.

The computers we have in mind have a distributed memory architecture with powerful processing nodes supporting the MIMD programming model. Concrete machines of that kind are the Intel Paragon, Thinking Machine's CM-5, and workstation clusters. The architecture of such machines makes programming with a coarse grain parallelism necessary for optimal exploitation of the compute power.

On serial computers, Gaussian elimination, with partial pivoting if necessary, is the method of choice for solving (1.1). We will review parallel algorithms used on shared memory multiprocessor computers such as Cray or Convex machines and investigate their scalability properties and thus suitability for implementation on distributed memory machines. Redundancies and speedups will be given with respect to Gaussian elimination.

We consider three approaches for solving (1.1) in parallel, divide and conquer in section 3, single-width separator in section 4, and double-width separator in section 5. The first approach is suited for diagonally dominant matrices, the second for diagonally dominant and symmetric definite matrices. The last approach permits partial pivoting in a natural way and is therefore suited for arbitrary (non-singular) matrices. In section 6 we compare the three methods and discuss a numerical implementation of the most promising one.

2 Gaussian elimination for band matrices

Gaussian elimination is the method of choice for solving (1.1) on serial computers [9, §4.3]. It consist of three phases, LU factorization, forward and then backward substitution. In the first phase the matrix A is factored into the product of a lower triangular matrix L and an upper triangular matrix U, $A = LU$. L has lower half-bandwidth r and U has upper half-bandwidth s. If partial row pivoting is necessary when factoring A, the upper half-bandwidth of U becomes $r + s$.

The computational complexity of the LU factorization of A without pivoting is

$$C_{\mathrm{LU}}(n, r, s) = (2s + 1)rn - r(r + 1)/2 - rs(r + 1) - s(s^2 - 1)/3 \text{ flops};$$

with pivoting it increases in the worst case, i.e. if the all pivots are located on the lowest off-diagonal of A, to

$$C_{\mathrm{LUP}}(n, r, s) = (2r + 2s + 1)rn - r(r + 1)(8r + 1)/6 - rs(2r + s + 1) \text{ flops}.$$

We measure complexities in flop, i.e. floating point operations. A flop is either an addition, a subtraction, a multiplication, or a division. For simplicity, we also count a square root as one flop.

Forward substitution with a banded lower triangular matrix with half-bandwidth r and backward substitution with a banded upper triangular matrix with half-bandwidth s cost

$$C_{\mathrm{forw}}(n, r) = 2nr - r(r + 1) \text{ flops} \qquad \text{and} \qquad C_{\mathrm{back}}(n, s) = (2s - 1)n - s(s + 1) \text{ flops},$$

respectively. The complexities of Gaussian elimination without and with pivoting are

$$C_{\mathrm{Gauss}}(n, r, s) = C_{\mathrm{LU}}(n, r, s) + C_{\mathrm{forw}}(n, r) + C_{\mathrm{back}}(n, s) \tag{2.1}$$

and

$$C_{\mathrm{GaussP}}(n, r, s) = C_{\mathrm{LUP}}(n, r, s) + C_{\mathrm{forw}}(n, r) + C_{\mathrm{back}}(n, r + s), \tag{2.2}$$

respectively. In the complexity analyses of the parallel algorithms we will restrict ourselves to the two important special cases

1. A is diagonally dominant and its two half-bandwidths are equal, $k := r = s$. Then,

$$C_{\mathrm{Gauss}}(n, k) = (2k^2 + 5k - 1)\, n - \frac{1}{6}k(8k + 13)(k + 1) \text{ flops}. \tag{2.3}$$

2. A is symmetric positive definite (spd) with half-bandwidth k. Then, instead of the LU factorization, we compute the Cholesky factorization, $A = LL^T$. This factorization halves the cost of solving $A\mathbf{x} = \mathbf{b}$,

$$C_{\mathrm{Cholesky}}(n, k) = (k^2 + 6k - 1)\, n - \frac{1}{6}k(4k + 17)(k + 1) \text{ flops}. \tag{2.4}$$

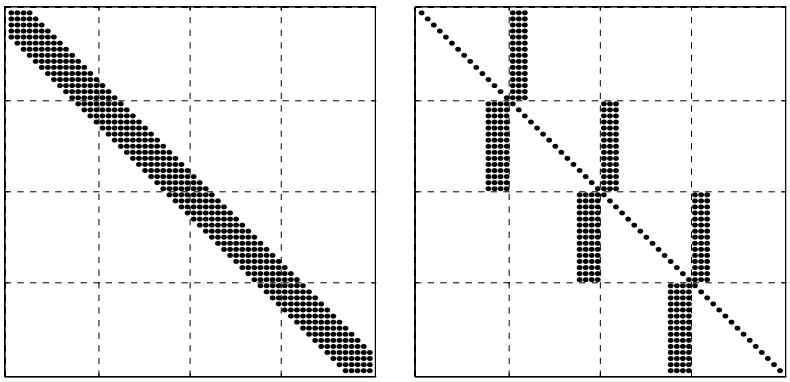
3 The divide and conquer approach

The divide and conquer approach we present here has been investigated by Dongarra and Sameh [7] for non-symmetric diagonally dominant band matrices. Earlier, Lawrie and Sameh [14] considered the same algorithm in connection with symmetric positive definite band matrices in which case the implementation is simplified and the complexity is reduced due to symmetry. The divide and conquer algorithm of Bondeli [3] for the solution of tridiagonal linear systems is a special case of the Dongarra-Sameh algorithm.

In the divide and conquer approach the matrix A and the vectors $\mathbf{x}$ and $\mathbf{b}$ are partitioned in the form

$$\begin{pmatrix} A_1 & B_1 & & & \\ C_2 & A_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & B_{p-1} \\ & & & C_p & A_p \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{p-1} \\ \mathbf{b}_p \end{pmatrix}, \tag{3.1}$$

where $A_i \in \mathbb{R}^{n_i \times n_i}$, $\mathbf{x}_i$, $\mathbf{b}_i \in \mathbb{R}^{n_i}$, $\sum_{i=1}^{p} n_i = n$. We assume that $n_i > r + s$. The structure of A and its submatrices is depicted in Fig. 3.1a for the case $p = 3$. Clearly, the diagonal blocks A_i are band matrices with the same half-bandwidths as A itself.



(a) (b)

Figure 3.1: Non-zero structure of (a) the matrix in (3.1) and (b) the matrix in (3.2) with $n = 60$, $p = 4$, $n_i = 15$, $r = 4$, and $s = 3$. Dots denote non-zeros.

We assume that we have p processors available the ith of which deals with the ith block row in (3.1); the matrices A_i, B_i, C_i, and the vectors $\mathbf{x}_i$ and $\mathbf{b}_i$ reside in the local memory of the i-th processor. Of the matrices $B_i \in \mathbb{R}^{n_i \times n_{i+1}}$, $i < p$, and $C_i \in \mathbb{R}^{n_{i-1} \times n_i}$, $i > 1$, only those columns are stored which contain nonzero elements. If we denote the first s columns of B_i by $\hat{B}_i$ and the last r columns of C_i by $\hat{C}_i$, then $C_i = [O_{n_i \times n_{i-1}-r}, \hat{C}_i]$ and $B_i = [\hat{B}_i, O_{n_i \times n_{i+1}-s}]$.

The algorithm

The algorithm proceeds in three steps.

1. Factorization

The submatrices A_i inherit the diagonal dominance of A and are, by consequence, invertible. We thus can multiply (3.1) from the left by $(A_1 \oplus \cdots \oplus A_p)^{-1}$ and obtain

$$\begin{pmatrix} I_{n_1} & E_1 & & & \\ F_2 & I_{n_2} & E_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & E_{p-1} \\ & & & F_p & I_{n_p} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \end{pmatrix} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{p-1} \\ \mathbf{c}_p \end{pmatrix}, \tag{3.2}$$

where $E_i = A_i^{-1} B_i$, $F_i = A_i^{-1} C_i$, and $\mathbf{c}_i = A_i^{-1} \mathbf{b}_i$. The structure of the matrix in (3.2) is depicted in Fig. 3.1b.

The A_i can be factored in parallel by Gaussian elimination without pivoting, $A_i = L_i U_i$. At the same time the s first columns of the matrices E_i and the last r columns of the F_i,

$$\hat{E}_i = U_i^{-1} L_i^{-1} \hat{B}_i, \qquad \hat{F}_i = U_i^{-1} L_i^{-1} \hat{C}_i,$$

and $\mathbf{c}_i = U_i^{-1} L_i^{-1} \mathbf{b}_i$ can be computed.

If we assume $k = r = s$ and $n_i = n/p$ for all i, then the *serial* complexity of this step is

$$\begin{aligned} C_{\text{step\_1}} &= pC_{\text{LU}}(\frac{n}{p}, k) + (p-1)\frac{k(k^2-1)}{3} + ((p-1)k + p)C_{\text{forw}}(\frac{n}{p}, k) + ((p-1)2k + p)C_{\text{back}}(\frac{n}{p}, k) \\ &= \left((8k^2 + 3k - 1) - \frac{k}{p}(6k - 2)\right) n - \frac{kp}{2}(k+1)(8k+5) + \frac{k}{3}(k+1)(8k+1) \text{ flops.} \end{aligned}$$

The term $k(k^2-1)/3$ stems from the computation of $L_i^{-1}\hat{B}_i$, which preserves the sparsity structure of $\hat{B}_i$.

In the factorization step, each processor can work independently on its block row without interprocessor communication. Therefore, the *parallel* complexity of this step is

$$\begin{aligned} C_{\text{step\_1}}^{\text{par}} &= C_{\text{LU}}(\frac{n}{p},k) + \frac{k}{3}(k^2-1) + (k+1)C_{\text{forw}}(\frac{n}{p},k) + (2k+1)C_{\text{back}}(\frac{n}{p},k) \\ &= (8k^2+3k-1)\frac{n}{p} - \frac{1}{2}k(k+1)(8k+5) \text{ flops.} \end{aligned}$$

The work in this step is not completely balanced, as processors 1 and p have only one off-diagonal block to compute. However, with large p, the effect of this imbalance becomes negligible.

2. Formation and solution of reduced system

The formation and solution of the reduced system is the difficult and crucial step of all the algorithms we consider in this paper. This step comprises the links between the split subsystems and therefore results in communication between processors.

From Fig. 3.1b we see that the last r equations of the the block rows 1 to $p-1$ and the first s equations of the block rows 2 to p in (3.2) determine the last r components of the $\mathbf{x}_i$, $1 \le i < p$, and the first s components of the $\mathbf{x}_i$, $2 \le i \le p$, respectively. More precisely, let $\hat{E}_i$, $\hat{F}_i$, $\mathbf{x}_i$, and $\mathbf{c}_i$ be partitioned into their first s, middle $n_i - r - s$, and last r rows,

$$\hat{E}_i = \begin{bmatrix} \hat{E}_{i1} \\ \hat{E}_{i2} \\ \hat{E}_{i3} \end{bmatrix}, \qquad \hat{F}_i = \begin{bmatrix} \hat{F}_{i1} \\ \hat{F}_{i2} \\ \hat{F}_{i3} \end{bmatrix}, \qquad \mathbf{x}_i = \begin{bmatrix} \mathbf{x}_{i1} \\ \mathbf{x}_{i2} \\ \mathbf{x}_{i3} \end{bmatrix}, \qquad \mathbf{c}_i = \begin{bmatrix} \mathbf{c}_{i1} \\ \mathbf{c}_{i2} \\ \mathbf{c}_{i3} \end{bmatrix}. \tag{3.3}$$

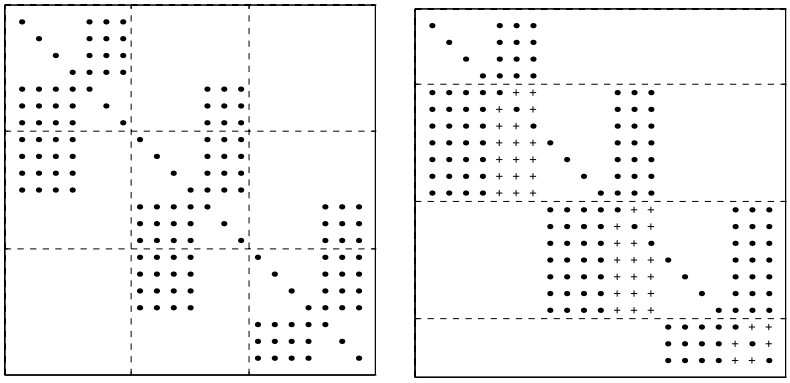
Then, extracting from (3.2) the above mentioned equations yields a *reduced* linear system

$$\underbrace{\left[\begin{array}{cc|cc|cc} I_r & \hat{E}_{13} & & & & \\ \hat{F}_{21} & I_s & O_{s\times r} & \hat{E}_{21} & & \\ \hline \hat{F}_{23} & O_{r\times s} & I_r & \hat{E}_{23} & \ddots & \\ & & \hat{F}_{31} & I_s & & \\ \hline & & \ddots & & \ddots & O_{s\times r} \quad \hat{E}_{p-1,1} \\ \hline & & & & \hat{F}_{p-1,3} \quad O_{r\times s} & I_r \quad \hat{E}_{p-1,3} \\ & & & & & \hat{F}_{p1} \quad I_s \end{array}\right]}_{S} \underbrace{\begin{bmatrix} \mathbf{x}_{13} \\ \mathbf{x}_{21} \\ \mathbf{x}_{23} \\ \mathbf{x}_{31} \\ \vdots \\ \mathbf{x}_{p-1,1} \\ \mathbf{x}_{p-1,3} \\ \mathbf{x}_{p,1} \end{bmatrix}}_{\mathbf{z}} = \underbrace{\begin{bmatrix} \mathbf{c}_{13} \\ \mathbf{c}_{21} \\ \mathbf{c}_{23} \\ \mathbf{c}_{31} \\ \vdots \\ \mathbf{c}_{p-1,1} \\ \mathbf{c}_{p-1,3} \\ \mathbf{c}_{p,1} \end{bmatrix}}_{\mathbf{h}} \tag{3.4}$$

of order $(p-1)(r+s)$. As indicated in Fig. 3.2a, the system matrix S is block-tridiagonal with blocks of order $r+s$. Fig. 3.2b shows how S can be formed in parallel. The dots between the dashed lines indicate the non-zero elements that are formed by the same processor.

As S inherits diagonal dominance from A, *Gaussian elimination without pivoting* is the straightforward method to solve $S\mathbf{z} = \mathbf{h}$. Factorization and forward substitution are performed sequentially: processor 1 starts by sending its r rows of the system to processor 2. This prepends the received to its own $r+s$ rows and eliminates the first $r+s$ equations of its local $(2r+s) \times ((2r+s)$ system. It then sends its (updated) last r rows to processor 3, and so on. Finally, processor p concludes by solving a system of order $r+s$. Fig. 3.2b shows the fill-in elements (+) that are produced during the factorization. Back-substitution then proceeds from processor p back to processor 1.

The volume of the messages sent from processor i to processor $i+1$ during the factorization phase are $r(s+1)$ floating point numbers, the elements of $\hat{E}_{i3}$ and $\mathbf{c}_{i3}$; in the backsubstitution phase the messages that are sent by processor i to processor $i-1$ have length s. They comprise $\mathbf{x}_{i+1,1}$.



(a) (b)

Figure 3.2: Two views of the reduced system matrix for $p = 4, r = 4, s = 3$. In (a) its block-tridiagonal structure in (b) its locality and fill-in (+).

The serial complexity of the solution of the reduced system is

$$C_{\text{step\_2}} = \frac{k}{6}(28k^2 + 45k - 1)(p-1) - k(4k^2 + 4k - 1) \text{ flops},$$

when we take the special structure of S, in particular its unit diagonal, into account.

The complexity of the *communication* is

$$2(p-1)\sigma + (p-1)(k^2 + 2k)\tau \text{ flops}.$$

Here, we assumed that the time for the transmission of a message of length n floating point numbers from one to another processor is of the form

$$\sigma + n\tau.$$

σ denotes the startup time *relative* to the time of a floating point operation i.e. the number of flops that can be executed during the startup time. τ denotes the number of floating point operations that can be executed during the transmission of one (8-Byte) floating point number. The parallel complexity of this step differs from the sequential complexity only in that includes the communication complexity. Thus,

$$C^{\text{par}}_{\text{step\_2}} = (p-1)\left(\frac{k}{6}(28k^2 + 45k - 1) + 2\sigma + (k^2 + 2k)\tau\right) - k(4k^2 + 4k - 1) \text{ flops}.$$

For an Intel Paragon running the operating system OSF/1 Release 1.2 the startup time for message passing is about 65μsec. The interprocessor communication bandwidth is about 65MB/sec. The 10 Mflop performance of the LINPACK benchmark [5] reflects the attainable performance for a program that doesn't make use of the pipelining features of the i860XP processor. Therefore, we set the numbers $\sigma = 1000$ and $\tau = 1$.

A second possibility for solving the reduced system is *block cyclic reduction* [9, p. 173]. Let the block-tridiagonal system (3.4) be written as

$$\begin{pmatrix} T_1 & U_1 & & & \\ V_2 & T_2 & U_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & U_{p-2} \\ & & & V_{p-1} & T_{p-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{\zeta}_1 \\ \boldsymbol{\zeta}_2 \\ \vdots \\ \boldsymbol{\zeta}_{p-2} \\ \boldsymbol{\zeta}_{p-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\eta}_1 \\ \boldsymbol{\eta}_2 \\ \vdots \\ \boldsymbol{\eta}_{p-2} \\ \boldsymbol{\eta}_{p-1} \end{pmatrix}. \tag{3.5}$$

To describe the algorithm, we imagine (3.5) being odd-even permuted into the form

$$\left[\begin{array}{cccccc|cccccc}
T_1 & & & & & & U_1 & & & & & \\
& T_3 & & & & & V_3 & U_3 & & & & \\
& & \ddots & & & & & \ddots & \ddots & & & \\
& & & T_{p_1-2} & & & & & \ddots & U_{p_1-2} & & \\
& & & & T_{p_1} & & & & & V_{p_1} & & \\
\hline
V_2 & U_2 & & & & & T_2 & & & & & \\
& V_4 & U_4 & & & & & T_4 & & & & \\
& & \ddots & \ddots & & & & & \ddots & & & \\
& & & V_{p_2} & U_{p_2} & & & & & T_{p_2} & &
\end{array}\right]
\left[\begin{array}{c}
\zeta_1 \\ \zeta_3 \\ \vdots \\ \zeta_{p_1-2} \\ \zeta_{p_1} \\ \hline \zeta_2 \\ \zeta_4 \\ \vdots \\ \zeta_{p_2}
\end{array}\right] =
\left[\begin{array}{c}
\eta_1 \\ \eta_3 \\ \vdots \\ \eta_{p_1-2} \\ \eta_{p_1} \\ \hline \eta_2 \\ \eta_2 \\ \vdots \\ \eta_{p_2}
\end{array}\right] \tag{3.6}$$

with $p_2 = p-1 = p_1+1$ if $p-1$ even, and $p_1 = p-1 = p_2+1$ if $p-1$ odd. ((3.6) shows the latter case.) Equation (3.6) is now block-factored, the left hand factor being applied to the right hand side immediately,

$$\left[\begin{array}{cccccc|cccccc}
I & & & & & & T_1^{-1}U_1 & & & & & \\
& I & & & & & T_3^{-1}V_3 & T_3^{-1}U_3 & & & & \\
& & \ddots & & & & & \ddots & \ddots & & & \\
& & & I & & & & & \ddots & T_{p_1-2}^{-1}U_{p_1-2} & & \\
& & & & I & & & & & T_{p_1}^{-1}V_{p_1} & & \\
\hline
& & & & & & T_2 & U_2 & & & & \\
& & & & & & \hat{V}_4 & \hat{T}_4 & \ddots & & & \\
& & & & & & & \ddots & \ddots & \hat{U}_{p_2-2} & & \\
& & & & & & & & \hat{V}_{p_2} & \hat{T}_{p_2} &
\end{array}\right]
\left[\begin{array}{c}
\zeta_1 \\ \zeta_3 \\ \vdots \\ \zeta_{p_1-2} \\ \zeta_{p_1} \\ \hline \zeta_2 \\ \zeta_4 \\ \vdots \\ \zeta_{p_2}
\end{array}\right] =$$

$$= \left[\begin{array}{cccccc|cccc}
T_1 & & & & & & & & & \\
& T_3 & & & & & & & & \\
& & \ddots & & & & & & & \\
& & & T_{p_1-2} & & & & & & \\
& & & & T_{p_1} & & & & & \\
\hline
V_2 & U_2 & & & & & I & & & \\
& V_4 & U_4 & & & & & I & & \\
& & \ddots & \ddots & & & & & \ddots & \\
& & & V_{p_2} & U_{p_2} & & & & & I
\end{array}\right]^{-1}
\left[\begin{array}{c}
\eta_1 \\ \eta_3 \\ \vdots \\ \eta_{p_1-2} \\ \eta_{p_1} \\ \hline \eta_2 \\ \eta_2 \\ \vdots \\ \eta_{p_2}
\end{array}\right] =
\left[\begin{array}{c}
T_1^{-1}\eta_1 \\ T_3^{-1}\eta_3 \\ \vdots \\ T_{p_1-2}^{-1}\eta_{p_1-2} \\ T_{p_1}^{-1}\eta_{p_1} \\ \hline \hat{\eta}_2 \\ \hat{\eta}_4 \\ \vdots \\ \hat{\eta}_{p_2}
\end{array}\right], \tag{3.7}$$

where

$$\begin{aligned}
\hat{T}_k &= T_k - U_k T_{k-1}^{-1} V_{k-1} - V_k T_{k+1}^{-1} U_{k+1}, \\
\hat{V}_k &= -V_k T_{k+1}^{-1} V_{k+1}, \qquad \hat{U}_k = -U_k T_{k+1}^{-1} U_{k+1}, \\
\hat{\eta}_k &= \eta_k - V_k T_{k-1}^{-1} \eta_{k-1} - U_k T_{k+1}^{-1} \eta_{k+1}.
\end{aligned}$$

The T_k, U_k, V_k, and η_k are stored in the memory of processor k. Therefore, the odd-numbered processors can compute $T_k^{-1}U_k$, $T_k^{-1}V_k$, and $T_k^{-1}\eta_k$. They then send these quantities to their neighboring even-numbered processors which compute their copies of $\hat{T}_k$, $\hat{U}_k$, $\hat{V}_k$, and $\hat{\eta}_k$. The even-numbered processors then block-factor the system

$$\left[\begin{array}{cccc}
\hat{T}_2 & \hat{U}_2 & & \\
\hat{V}_4 & \hat{T}_4 & \ddots & \\
& \ddots & \ddots & \hat{U}_{p_2-2} \\
& & \hat{V}_{p_2} & \hat{T}_{p_2}
\end{array}\right]
\left[\begin{array}{c}
\zeta_2 \\ \zeta_4 \\ \vdots \\ \zeta_{p_2}
\end{array}\right] =
\left[\begin{array}{c}
\hat{\eta}_2 \\ \hat{\eta}_4 \\ \vdots \\ \hat{\eta}_{p_2}
\end{array}\right] \tag{3.8}$$

in the same way as before. This recursive process is continued until a $2k \times 2k$ system of equations is left, which is then solved directly. The other components of $\mathbf{z}$ are obtained by back-substitution.

Interestingly, the nonzero structures of $\hat{T}_k$, $\hat{U}_k$, and $\hat{V}_k$ are the same as those of T_k, U_k, and V_k, respectively! Because the nonzero patterns are preserved during the cyclic reduction, the sequential and parallel complexities are given by

$$C_{\text{step\_2,cr}} \lesssim (p-1)\frac{k}{6}\left(124k^2+69k-7\right) \text{ flops}$$

and

$$C^{\text{par}}_{\text{step\_2,cr}} \lesssim \log_2\lfloor p-1\rfloor\left(\frac{k}{6}\left(124k^2+69k-7\right)+4\sigma+(4k^2+4k)\tau\right)+\frac{1}{6}k(16k^2+33k-7) \text{ flops},$$

respectively.

3. Back substitution

Equation (3.4) determines some of the first and last components of the $\mathbf{x}_i$, $1 \le i \le p$. The undetermined components of the $\mathbf{x}_i$ are computed by back-substitution. (3.2) with the notation of (3.3) gives

$$\begin{aligned}
\begin{bmatrix} \mathbf{x}_{11} \\ \mathbf{x}_{12} \end{bmatrix} &= \begin{bmatrix} \mathbf{c}_{11} \\ \mathbf{c}_{12} \end{bmatrix} - \begin{bmatrix} \hat{E}_{11} \\ \hat{E}_{12} \end{bmatrix} \mathbf{x}_{21}, \\
\mathbf{x}_{i2} &= \mathbf{c}_{i2} - \hat{F}_{i2}\mathbf{x}_{i-1,3} - \hat{E}_{i2}\mathbf{x}_{i+1,1}, \qquad 1 < i < p, \\
\begin{bmatrix} \mathbf{x}_{p2} \\ \mathbf{x}_{p3} \end{bmatrix} &= \begin{bmatrix} \mathbf{c}_{p2} \\ \mathbf{c}_{p3} \end{bmatrix} - \begin{bmatrix} \hat{F}_{p2} \\ \hat{F}_{p3} \end{bmatrix} \mathbf{x}_{p-1,3}.
\end{aligned} \tag{3.9}$$

As the first, the third step of this algorithm is trivial to parallelize. There is no interprocessor communication. So, the serial and parallel complexities of this step are

$$C_{\text{step\_3}} = 4k\left(1-\frac{1}{p}\right)n - 8k^2p + 12k^2 \text{ flops} \qquad \text{and} \qquad C^{\text{par}}_{\text{step\_3}} = 4k\frac{n}{p} - 8k^2 \text{ flops},$$

respectively. Notice that the work of this step is much smaller than that of step 1.

Redundancy and speedup

The discussion in this section is made under the assumption that $k := r = s \ll n$. We first consider the variant of this divide and conquer algorithm in which the reduced system is solved by block Gaussian elimination. The overall *serial* complexity of this algorithm is

$$C_{\text{d\&c}}(n,k,p) = C_{\text{step\_1}} + C_{\text{step\_2}} + C_{\text{step\_3}} \approx \left(8k^2 - 6k^2\frac{1}{p}\right)n + \frac{2}{3}pk^3 \text{ flops}. \tag{3.10}$$

This is clearly bigger than the complexity of the ordinary Gaussian elimination (2.1). The ratio of the two complexities is called *redundancy* [12, p. 113]. As $k \ll n$ we omit the terms depending only on k in $C_{\text{step\_3}}$, $C_{\text{step\_3}}$, and C_{Gauss}. Thus,

$$R_{\text{d\&c}}(n,k,p) := \frac{C_{\text{d\&c}}(n,k,p)}{C_{\text{Gauss}}(n,k)} \approx 4 - \frac{3}{p} + \frac{1}{3}\frac{pk}{n} \tag{3.11}$$

Notice, that $\frac{pk}{n} \le 1$. Redundancy measures the overhead work that is introduced by parallelizing an algorithm. In this algorithm it is caused by the computation of the matrices $\hat{E}_i$ and $\hat{F}_i$ in step 1.

Speedup is the factor by which a parallel algorithm on p processors is faster than the best sequential algorithm. As the *parallel* complexity of the divide and conquer algorithm is

$$C^{\text{par}}_{\text{d\&c}} = C^{\text{par}}_{\text{step\_1}} + C^{\text{par}}_{\text{step\_2}} + C^{\text{par}}_{\text{step\_3}} \approx 8k^2\frac{n}{p} + \left(\frac{14}{3}k^3 + 2\sigma + k^2\tau\right)p \text{ flops}, \tag{3.12}$$

speedup becomes

$$S_{\mathrm{d\&c}}(n,k,p) := \frac{C_{\mathrm{Gauss}}(n,k)}{C^{\mathrm{par}}_{\mathrm{d\&c}}(n,k,p)} \approx \frac{p}{4+\left(\frac{7}{3}k+\frac{1}{2}\tau+\frac{\sigma}{k^2}\right)\frac{p^2}{n}} \tag{3.13}$$

Clearly, this speedup is far from the ideal speedup $S(p)=p$. As

$$S_{\mathrm{d\&c}}(n,k,p) = \frac{C_{\mathrm{Gauss}}(n,k)}{C^{\mathrm{par}}_{\mathrm{d\&c}}(n,k,p)} = \frac{C_{\mathrm{Gauss}}(n,k)}{C_{\mathrm{d\&c}}(n,k,p)}\,\frac{C_{\mathrm{d\&c}}(n,k,p)}{C^{\mathrm{par}}_{\mathrm{d\&c}}(n,k,p)} \leq \frac{p}{R(n,k,p)},$$

ideal speedup is possible only if (1) the parallel algorithm has the same complexity as the sequential algorithm and (2) if the parallel algorithm is perfectly parallelizable. This algorithm satisfies neither of the two conditions. First, the redundancy is very high, $R\approx 4$, and, second, the reduced system is solved serially. The latter is the origin of the p^2 term in (3.13) which makes speedup *decrease* as soon as p exceeds a critical number $p^{\mathrm{opt}}_{\mathrm{d\&c}}$ for which speedup is highest. $p^{\mathrm{opt}}_{\mathrm{d\&c}}$ is the positive zero of the derivative of S in (3.13),

$$p^{\mathrm{opt}}_{\mathrm{d\&c}} \approx \sqrt{\frac{12n}{7k}}\sqrt{\frac{1}{1+\frac{3\tau}{14k}+\frac{3\sigma}{7k^3}}}, \tag{3.14}$$

and yields an optimal speedup of

$$S^{\mathrm{opt}}_{\mathrm{d\&c}} := S(n,k,p_{\mathrm{opt}}) \approx \sqrt{\frac{3n}{112k}}\sqrt{\frac{1}{1+\frac{3\tau}{14k}+\frac{3\sigma}{7k^3}}} = \frac{1}{8}p^{\mathrm{opt}}_{\mathrm{d\&c}}. \tag{3.15}$$

The serial complexity of the variant of the divide and conquer algorithm in which the reduced system is solved by cyclic reduction is

$$C_{\mathrm{d\&c,cr}}(n,k,p) = C_{\mathrm{step\_1}} + C_{\mathrm{step\_2,cr}} + C_{\mathrm{step\_3}} \approx \left(8k^2 - 6k^2\frac{1}{p}\right)n + \frac{50}{3}pk^3 \text{ flops} \tag{3.16}$$

whereas the parallel complexity is

$$C^{\mathrm{par}}_{\mathrm{d\&c,cr}} = C^{\mathrm{par}}_{\mathrm{step\_1}} + C^{\mathrm{par}}_{\mathrm{step\_2,cr}} + C^{\mathrm{par}}_{\mathrm{step\_3}} \approx 8k^2\frac{n}{p} + \left(\frac{62}{3}k^3 + 4\sigma + 8k^2\tau\right)\log_2(p) \text{ flops}. \tag{3.17}$$

From this we get the redundancy

$$R_{\mathrm{d\&c,cr}}(n,k,p) := \frac{C_{\mathrm{d\&c,cr}}(n,k,p)}{C_{\mathrm{Gauss}}(n,k)} \approx 4 - \frac{3}{p} + \frac{25}{3}\frac{pk}{n}. \tag{3.18}$$

Not surprisingly, since the factor 4 stems from step 1, this is not much different from (3.11). The speedup is given by

$$S_{\mathrm{d\&c,cr}}(n,k,p) := \frac{C_{\mathrm{Gauss}}(n,k)}{C^{\mathrm{par}}_{\mathrm{d\&c,cr}}(n,k,p)} \approx \frac{p}{4+\left(\frac{31}{3}k+4\tau+2\frac{\sigma}{k^2}\right)\frac{p\log_2 p}{n}}. \tag{3.19}$$

Here, optimal processor number and optimal speedup are

$$p^{\mathrm{opt}}_{\mathrm{d\&c,cr}} \approx \frac{12n}{31k}\,\frac{1}{1+\frac{12\tau}{31k}+\frac{6\sigma}{31k^3}}, \tag{3.20}$$

and

$$S^{\mathrm{opt}}_{\mathrm{d\&c,cr}} \approx \frac{3n}{31k+12\tau+6\sigma/k^2}\,\frac{1}{1+\log_2\left(\frac{12n}{31k+12\tau+6\sigma/k^2}\right)} = \frac{1}{4}p^{\mathrm{opt}}_{\mathrm{d\&c,cr}}\frac{1}{1+\log_2(p^{\mathrm{opt}}_{\mathrm{d\&c,cr}})}. \tag{3.21}$$

For large ratios n/k these optimal quantities are much higher than (3.14) and (3.15).

Memory requirements

The matrices $\hat{E}_i$ and $\hat{F}_i$ can be stored in A_i. Extra memory space is needed to store the reduced system. If the reduced system is solved by cyclic reduction, space for about $16k^2$ floating point numbers is needed. If the reduced system is solved by Gaussian elimination, only about half of this is needed. If the reduced system is solved on a single processor, the complete reduced system must be stored there.

Remarks

Notice that it is *not* sufficient to multiply (3.1) by $(L_1 \oplus \cdots \oplus L_p)^{-1}$ from the left. This does not lead to a reduced system.

The reduced system is *not* symmetric even if the original matrix is so! If A is positive definite, the A_i can be factored according to Cholesky which saves only $k^2n - \frac{2}{3}pk^3$ flops of the $8k^2n$ flops of the overall algorithm. The reduced system does not have a Cholesky factorization in general.

Mehrmann [15] studies the case where the A_i are not diagonally dominant or even non-singular. He proposes writing the diagonal blocks in the form $A_i = \tilde{A}_i + D_i$ where $\tilde{A}_i$ equals A_i up to the first and last $k \times k$ principal diagonal blocks. D_i is chosen so that $\tilde{A}_i$ is non-singular. There are free parameters so that the condition of $\tilde{A}_i$ could be optimized. The value of such a procedure is questionable, since the complexity of the solution of the reduced system is increased considerably as the particular structure (3.4) is lost. We believe that it is easier to solve such systems by the double-width separator approach that will be introduced in section 5.

Meier [16] introduces a band solver which is similar to the method proposed by Wang for tridiagonal systems of equations.

The so far mentioned papers did not consider solving the reduced system by cyclic reduction. The authors implemented the methods on computers with only a few processors. However, for cyclic reduction to be superior over Gaussian elimination the number p of processors has to be sufficiently large.

4 The single-width separator approach

In contrast to the divide and conquer approach, in the single-width separator approach there is a layer between the diagonal blocks. The width of the layer equals the larger of the half-bandwidths. This algorithm has been studied e.g. by Johnsson [13], Dongarra and Johnsson [6], and in modified forms by Wright [17], Conroy [4]. It is well-suited for diagonally dominant and symmetric definite matrices. For the presentation we assume A to be nonsymmetric diagonally dominant.

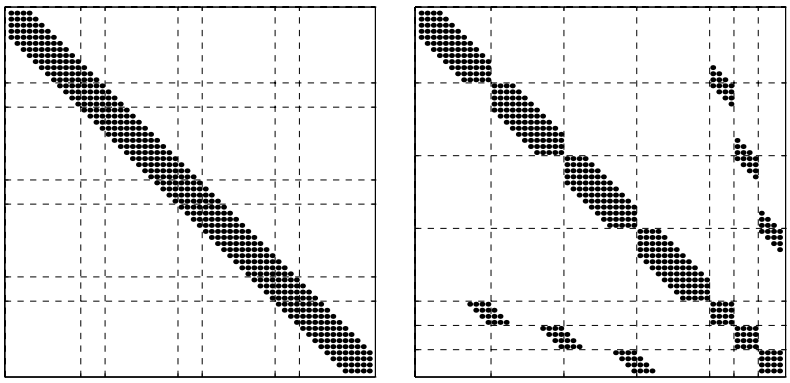
In the single-width separator approach the matrix A and the vectors $\mathbf{x}$ and $\mathbf{b}$ are partitioned in the form

$$\begin{pmatrix} A_1 & B_1 & & & & & \\ C_1 & D_1 & C_2 & & & & \\ & B_2 & A_2 & B_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & C_{2p-3} & D_{p-1} & C_{2p-2} & \\ & & & & B_{2p-2} & A_p & \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \boldsymbol{\xi}_1 \\ \mathbf{x}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \\ \mathbf{x}_p \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \boldsymbol{\beta}_1 \\ \mathbf{b}_2 \\ \vdots \\ \boldsymbol{\beta}_{p-1} \\ \mathbf{b}_p \end{pmatrix}, \tag{4.1}$$

where $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i \in \mathbb{R}^{n_i \times k}$, $C_i \in \mathbb{R}^{k \times n_i}$, $D_i \in \mathbb{R}^{k \times k}$, $\mathbf{x}_i$, $\mathbf{b}_i \in \mathbb{R}^{n_i}$, $\boldsymbol{\xi}_i$, $\boldsymbol{\beta}_i \in \mathbb{R}^k$, $k = \max(r, s)$, and $\sum_{i=1}^{p} n_i + (p-1)k = n$. We assume that $n_i > k$. The structure of A and its submatrices is depicted in Fig. 4.1a for the case $p = 4$. The diagonal blocks A_i are band matrices with the same half-bandwidths as A itself.

Remark. The following analysis also holds if we choose $k > \max(r, s)$. This however doesn't make sense practically, as the computational complexity grows with increasing k. The parallel portion of the algorithm decreases while the sequential portion increases. □

We again assume that we have p processors. In this case, processor i stores matrices A_i, B_{2i-2}, B_{2i-1}, C_{2i-2}, C_{2i-1}, D_i and the vectors $\mathbf{b}_i$ and $\boldsymbol{\xi}_i$. (Some of the variables may not be present on processor 1 and p.)



(a) (b)

Figure 4.1: Non-zero structure of (a) the original and (b) the block odd-even permuted band matrix with $n = 60$, $p = 4$, $n_i = 12$, $r = 4$, and $s = 3$.

The easiest way to understand the single-width separator approach is by permuting rows and columns of A in a block odd-even fashion. In this way, (4.1) becomes

$$\left[\begin{array}{cccccc|cccccc} A_1 & & & & & & B_1 & & & & & \\ & A_2 & & & & & B_2 & B_3 & & & & \\ & & \ddots & & & & & \ddots & \ddots & & & \\ & & & A_{p-1} & & & & & \ddots & B_{2p-3} & & \\ & & & & A_p & & & & & B_{2(p-1)} & & \\ \hline C_1 & C_2 & & & & & D_1 & & & & & \\ & C_3 & \ddots & & & & & D_2 & & & & \\ & & \ddots & \ddots & & & & & \ddots & & & \\ & & & C_{2p-3} & C_{2(p-1)} & & & & & & D_{p-1} & \end{array}\right] \left[\begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \\ \hline \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{array}\right] = \left[\begin{array}{c} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{p-1} \\ \mathbf{b}_p \\ \hline \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{p-1} \end{array}\right], \qquad (4.2)$$

the structure of which is depicted in Fig. 4.1b. We denote the matrix in (4.2) by $\tilde{A}$.

Remark. As matrices appearing in domain decomposition methods have the structure of $\tilde{A}$ the above permutation is sometimes called 'algebraic domain decomposition'. If $k = r$, the following factorization represents one step of cyclic reduction. □

The algorithm

Now, the algorithm proceeds in three steps.

1. Factorization

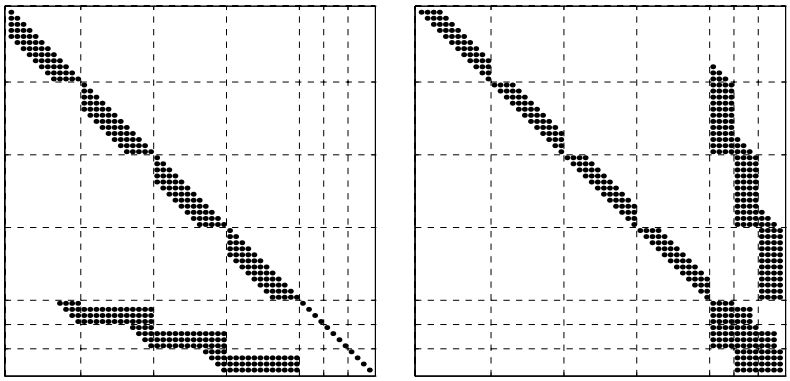
We compute a block LU factorization of the matrix in (4.2), $\tilde{A} = LR$. The structures of the L and R factors are depicted in Fig. 4.2. These are the same structures as George [8] obtained with the non-symmetric Cholesky factorization in his one-way dissection scheme. After multiplying (4.2)

by L^{-1}, we obtain the system

$$\left(\begin{array}{cccc|cccc} R_1 & & & & E_1 & & & \\ & R_2 & & & E_2 & E_3 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & & R_{p-1} & & & \ddots & E_{2p-3} \\ & & & & R_p & & & E_{2p-2} \\ \hline & & & & T_1 & U_1 & & \\ & & & & V_2 & T_2 & \ddots & \\ & & & & & \ddots & \ddots & U_{p-2} \\ & & & & & & V_{p-1} & T_{p-1} \end{array}\right) \left(\begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \\ \hline \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{array}\right) = \left(\begin{array}{c} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{p-1} \\ \mathbf{c}_p \\ \hline \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_{p-1} \end{array}\right) \tag{4.3}$$

where

$$\begin{aligned} &E_{2i-2} = L_i^{-1} B_{2i-2}, \quad E_{2i-1} = L_i^{-1} B_{2i-1}, \quad F_{2i-2} = C_{2i-2} R_i^{-1}, \quad F_{2i-1} = C_{2i-1} R_i^{-1}, \\ &\mathbf{c}_i = L_i^{-1} \mathbf{b}_i, \quad \boldsymbol{\gamma}_i = \boldsymbol{\beta}_i - F_{2i-1}\mathbf{c}_i - F_{2i}\mathbf{c}_{i+1}, \\ &T_i = D_i - F_{2i-1}E_{2i-1} - F_{2i}E_{2i}, \quad U_i = -F_{2i}E_{2i+1}, \quad V_i = -F_{2i-1}E_{2i-2}. \end{aligned} \tag{4.4}$$



(a) (b)

Figure 4.2: Non-zero structure of the (a) L and (b) R factor of the LU decomposition of $\tilde{A}$. Here, $n = 60$, $p = 4$, $n_i = 12$, $r = 4$, and $s = 3$.

Each processor can work independently on its block row computing E_{2i-2}, E_{2i-1}, F_{2i-2}, F_{2i-1}, and $\mathbf{c}_i$. Furthermore, each processor computes its portion of the matrix and right hand side of the reduced system (4.6),

$$\left[\begin{array}{cc} -F_{2i-2}E_{2i-2} & -F_{2i-2}E_{2i-1} \\ -F_{2i-1}E_{2i-2} & D_i - F_{2i-1}E_{2i-1} \end{array}\right] \in \mathbb{R}^{2k \times 2k} \qquad \text{and} \qquad \left[\begin{array}{c} -F_{2i-2}\mathbf{c}_i \\ \boldsymbol{\beta}_i - F_{2i-1}\mathbf{c}_i \end{array}\right] \in \mathbb{R}^{2k}, \tag{4.5}$$

respectively. Until this point of the algorithm, there is no interprocessor communication.

For the complexity analysis we assume that $k = r = s$ and that $n_i = n/p - k$. Thus, the *serial* complexity of this step is

$$\begin{aligned} C_{\text{step\_1}} &= pC_{\text{LU}}(\frac{n}{p}-k,k) + 2(p-1)\frac{k(k^2-1)}{3} + ((p-1)k+p)C_{\text{forw}}(\frac{n}{p}-k,k) \\ &\quad + (p-1)kC_{\text{back}}(\frac{n}{p}-k,k) + pk(\frac{2n}{p}-k+1) + (p-1)2k\left(\frac{k^2}{3}+\frac{k}{2}+\frac{1}{6}\right) + (p-2)2k^2(k+1) \\ &= \left((8k^2+4k) - \frac{k}{p}(6k-1)\right)n - \frac{kp}{2}(16k^2+9k+1) + \frac{k}{3}(8k^2-12k+1) \text{ flops.} \end{aligned}$$

The terms have been arranged in the same order as they appear in (4.4). Step 1 is almost perfectly parallelizable. Only the first and last processors have less work to do. The *parallel* complexity of this step is

$$\begin{aligned} C^{\rm par}_{\rm step\_1} &= C_{\rm LU}(\frac{n}{p},k) + 2\frac{k(k^2-1)}{3} + (k+1)C_{\rm forw}(\frac{n}{p}-k,k) \\ &\quad + kC_{\rm back}(\frac{n}{p}-k,k) + k(2\frac{n}{p}-k+1) + 2k\left(\frac{1}{3}k^2+\frac{1}{2}k+\frac{1}{6}\right) + 2k^2(k+1) \\ &= (8k^2+4k)\frac{n}{p} - \frac{k}{2}(16k^2+9k+1) \text{ flops.} \end{aligned}$$

2. Formation and solution of reduced system

The reduced system here is

$$S\boldsymbol{\xi} = \begin{pmatrix} T_1 & U_1 & & & \\ V_2 & T_2 & U_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & U_{p-2} \\ & & & V_{p-1} & T_{p-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-2} \\ \boldsymbol{\xi}_{p-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_{p-2} \\ \boldsymbol{\gamma}_{p-1} \end{pmatrix}. \tag{4.6}$$

The matrix S is the Schur complement of $\bigoplus_{i=1}^{p} A_i$ in $\tilde{A}$. It is a block tridiagonal matrix of order $(p-1)k$ with $k\times k$ blocks. These blocks may not be full if $r<k$ or $s<k$, cf. Fig 4.2b. Similarly as in the divide and conquer approach the reduced system is diagonally dominant and can be solved by block Gaussian elimination or by block cyclic reduction. In both cases, the matrices V_i, T_i, U_i and the vector $\boldsymbol{\gamma}_i$ are stored on processor $i-1$ at the beginning of the solution process.

Block Gaussian elimination of (4.6) costs

$$C_{\rm step\_2} = \frac{k}{6}(28k^2+27k-7)(p-1) - k^2(4k+3) \text{ flops.}$$

The complexity of the communication is $2(p-1)\sigma + (p-1)(k^2+k)\tau$ flops and thus the parallel complexity becomes

$$C^{\rm par}_{\rm step\_2} = (p-1)\left(\frac{k}{6}(28k^2+27k-7) + 2\sigma + k(k+2)\tau\right) - k^2(4k+3) \text{ flops.}$$

If the reduced system is solved by cyclic reduction, the serial and parallel complexities are given by

$$C_{\rm step\_2,cr} \lesssim (p-1)\frac{k}{6}\left(76k^2+21k-7\right) \text{ flops}$$

and

$$C^{\rm par}_{\rm step\_2,cr} \lesssim \log_2\lfloor p-1\rfloor\left(\frac{k}{6}\left(76k^2+21k-7\right) + 4\sigma + (4k^2+4k)\tau\right) + \frac{1}{6}k(4k^2+9k-7) \text{ flops,}$$

respectively.

3. Back substitution

Knowing the vectors $\boldsymbol{\xi}_i$, $1\le i<p$, the i-th processor can compute its section of $\mathbf{x}$ by

$$\begin{aligned} \mathbf{x}_1 &= R_1^{-1}(\mathbf{c}_1 - E_1\boldsymbol{\xi}_1), \\ \mathbf{x}_i &= R_i^{-1}(\mathbf{c}_i - E_{2i-2}\boldsymbol{\xi}_{i-1} - E_{2i-1}\boldsymbol{\xi}_i), \qquad 1<i<p, \\ \mathbf{x}_p &= R_p^{-1}(\mathbf{c}_p - E_{2p-2}\boldsymbol{\xi}_{p-1}). \end{aligned} \tag{4.7}$$

Each processor can proceed independently, there is no interprocessor communication. Therefore,

$$C_{\rm step\_3} = p\,C_{\rm back}(\frac{n}{p}-k,k) + 4(p-1)k(\frac{n}{p}-k) = \left(6k-1-\frac{4k}{p}\right)n - 7k^2p + 4k^2,$$

and

$$C^{\rm par}_{\rm step\_3} = (6k-1)\frac{n}{p} - 7k^2,$$

respectively.

Redundancy and speedup

We proceed as in the previous section. We assume that $k := r = s \ll n$. We first consider the variant of the single width separator algorithm in which the reduced system is solved by block Gaussian elimination. The overall *serial* complexity of this algorithm is

$$C_{\rm sws}(n,k,p) \approx \left(8k^2 - 6k^2\frac{1}{p}\right) n - \frac{10}{3}pk^3 \text{ flops.} \tag{4.8}$$

Comparing (4.8) with (2.3) we obtain the redundancy

$$R_{\rm sws}(n,k,p) := \frac{C_{\rm sws}(n,k,p)}{C_{\rm Gauss}(n,k)} \approx 4 - \frac{3}{p} - \frac{5}{3}\frac{pk}{n}, \tag{4.9}$$

which is almost as high as in the divide and conquer algorithm.

The significant terms of the *parallel* complexity of the single width separator algorithm and the divide and conquer algorithm are the same, cf. (3.12),

$$C^{\rm par}_{\rm sws} \approx 8k^2\frac{n}{p} + \left(\frac{14}{3}k^3 + 2\sigma + k^2\tau\right) p \text{ flops.} \tag{4.10}$$

Therefore, the speedup of divide and conquer and single width separator approach are equal.

The serial complexity of the variant of the single width separator algorithm in which the reduced system is solved by cyclic reduction is

$$C_{\rm sws,cr}(n,k,p) \approx \left(8k^2 - 6k^2\frac{1}{p}\right) n + \frac{14}{3}pk^3 \text{ flops,} \tag{4.11}$$

whereas the parallel complexity is

$$C^{\rm par}_{\rm sws,cr} \approx 8k^2\frac{n}{p} + \left(\frac{14}{3}k^3 + 4\sigma + 4k^2\tau\right)\log_2(p) \text{ flops.} \tag{4.12}$$

From this we get the redundancy

$$R_{\rm sws,cr}(n,k,p) := \frac{C_{\rm sws,cr}(n,k,p)}{C_{\rm Gauss}(n,k)} \approx 4 - \frac{3}{p} - \frac{7}{3}\frac{pk}{n}. \tag{4.13}$$

The speedup is given by

$$S_{\rm sws,cr}(n,k,p) := \frac{C_{\rm Gauss}(n,k)}{C^{\rm par}_{\rm sws,cr}(n,k,p)} \approx \frac{p}{4 + \left(\frac{7}{3}k + 2\tau + 2\frac{\sigma}{k^2}\right)\frac{p\log_2 p}{n}}. \tag{4.14}$$

Here, optimal processor number and optimal speedup are

$$p^{\rm opt}_{\rm sws,cr} \approx \frac{12n}{7k}\,\frac{1}{1 + \frac{6\tau}{7k} + \frac{6\sigma}{7k^3}}, \tag{4.15}$$

and

$$S^{\rm opt}_{\rm sws,cr} \approx \frac{3n}{7k + 6\tau + 6\sigma/k^2}\,\frac{1}{1 + \log_2\left(\frac{12n}{7k+6\tau+6\sigma/k^2}\right)} = \frac{1}{4}p^{\rm opt}_{\rm sws,cr}\frac{1}{1+\log_2(p^{\rm opt}_{\rm sws,cr})}, \tag{4.16}$$

respectively.

Memory requirements

The matrices E_i and F_i can be stored in A_i. The memory space needed for the local portions of the reduced system is about $4k^2$. This is less than in the divide and conquer algorithms as the blocks of the reduced system have only half the order.

Remarks

This algorithm is well suited for *symmetric, positive definite* systems of equations, as the factorization of (4.2) is essentially symmetric. In (4.4) we have $F_i = E_i^T$ and $V_{i+1} = U_i^T$. The amount of work is reduced by a factor of 2. From Section 2 we know that Cholesky factorization is cheaper than Gaussian elimination also by this factor. We compute the non-symmetric Cholesky factorization [8] which yields left and right factors with the same structures as those in Fig. 4.2. Also the volume of the messages is about half as large as in the non-symmetric case; however, their number remains the same. Therefore, the formulae for redundancy and speedup are not changed besides the weights of the terms involving the relative startup time σ. This holds for both variants of the algorithm.

Wright [17] presented a version of the single width separator algorithm with pivoting. His algorithm incorporates total pivoting. If the factorization of an A_i cannot be completed, the unfactored part is added to the reduced system. The programming of this algorithm is quite cumbersome and the load of the processors in a multiprocessor environment is unpredictable. The advantage of this algorithm is the combination of a good performance in favorable situations and a safeguard in the other cases.

Conroy [4] partitions the band matrix A as in (4.1), but his algorithm resembles more the divide and conquer algorithm of Section 3.

Johnsson's analysis in [13] is very similar to ours. He however models the communication complexity differently. There is no startup time. The time to communicate n floating point numbers is proportional to n and the *distance* between the communicating processors.

Cyclic reduction is practical only if a rich network as a hypercube or perfect shuffle network is available. On the newest generation MPP computers establishing links between processors does not play a significant role in the communication cost anymore. The startup time mainly includes the overhead due to initiation of a message transfer (system calls, buffer allocation). Thus, cyclic reduction is feasible on 2D grid connected processors.

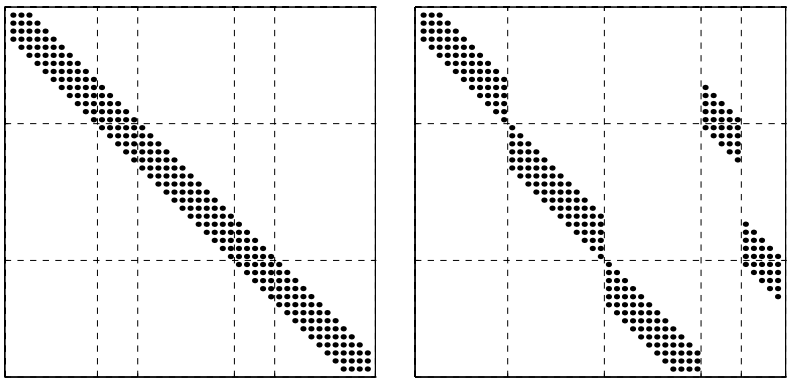
5 The double-width separator approach

In the double-width separator approach the separation layer is about twice as wide as in the single-width separator approach. The width m equals the sum of the two half-bandwidths, $m = r + s$. This algorithm has been studied by Wright [17] and Conroy [4], too. It permits partial pivoting in a natural way and thus is suited for systems of linear equations which are *not* positive definite or diagonally dominant.

In the double-width separator approach the matrix A and the vectors $\mathbf{x}$ and $\mathbf{b}$ are partitioned in the form (cf. Fig. 5.1a)

$$\begin{pmatrix} A_1 & B_1 & & & & & & & \\ & C_2 & A_2 & B_2 & & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & & C_{p-1} & A_{p-1} & B_{p-1} & \\ & & & & & & & C_p & A_p \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \boldsymbol{\xi}_1 \\ \mathbf{x}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \\ \mathbf{x}_p \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{p-1} \\ \mathbf{b}_p \end{pmatrix}, \qquad (5.1)$$

where $\mathbf{x}_i \in \mathbb{R}^{n_i}$, $\boldsymbol{\xi}_i \in \mathbb{R}^m$, and $\mathbf{b}_1 \in \mathbb{R}^{n_1+r}$, $\mathbf{b}_p \in \mathbb{R}^{n_p+s}$, $\mathbf{b}_i \in \mathbb{R}^{n_i+m}$, $1 < i < p$. The A_i are rectangular! Here, we have $\sum_{i=1}^{p} n_i + (p-1)m = n$.



(a) (b)

Figure 5.1: Non-zero structure of (a) the original and (b) the column permuted band matrix with $n = 45$, $p = 3$, $n_1 = 11$, $n_2 = n_3 = 12$, $r = 3$, and $s = 2$.

We permute the columns of the matrix A such that the separator columns are moved to the end (cf. Fig. 5.1b),

$$\begin{pmatrix} A_1 & & & & & B_1 & & & & \\ & A_2 & & & & C_2 & B_2 & & & \\ & & \ddots & & & & \ddots & \ddots & & \\ & & & A_{p-1} & & & & C_{p-1} & B_{p-1} & \\ & & & & A_p & & & & C_p & \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_p \\ \boldsymbol{\xi}_1 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{p-1} \\ \mathbf{b}_p \end{pmatrix}, \qquad (5.2)$$

We assume to have p processors available. The i-th processor holds A_i, B_i, C_i, and $\mathbf{b}_i$.

The algorithm

Also this algorithm proceeds in 3 steps.

1. Factorization

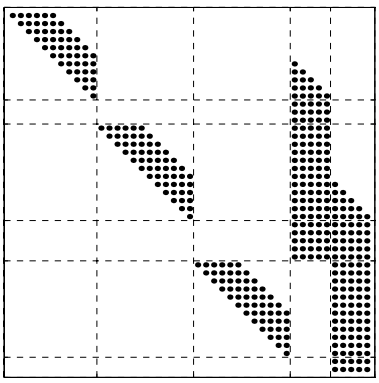


Figure 5.2: Non-zero structure of the R-factor

We compute the LU factorization $L_i R_i = P_i A_i$ of the rectangular A_i. P_i is a permutation matrix. Note that the A_i have full rank since A is nonsingular. We chose the maximum column

pivoting strategy to achieve stability. We apply $\bigoplus_{i=1}^{p}(L_i^{-1}P_i)$ on equation (5.2) from the left to obtain

$$\begin{pmatrix} R_1 & & & & E_1 & & & \\ & R_2 & & & F_2 & E_2 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & & R_{p-1} & & & F_{p-1} & E_{p-1} \\ & & & & R_p & & & F_p \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_p \\ \boldsymbol{\xi}_1 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \\ \mathbf{c}_{p-1} \\ \mathbf{c}_p \end{pmatrix}, \tag{5.3}$$

where

$$E_i = L_i^{-1}P_iB_i, \qquad F_i = L_i^{-1}P_iC_i, \qquad \mathbf{c}_i = L_i^{-1}P_i\mathbf{b}_i.$$

The structure of the matrix in (5.3) is depicted in Fig. 5.2. In most cases the fill-in in E_i and F_i is not as severe. Nevertheless, the memory has to be provided for this worst-case situation.

For the complexity analysis we assume for simplicity that the A_i are square and $n_i = n/p$. We furthermore assume that $k = r = s$. As the A_i are not assumed to be diagonally dominant, pivoting is necessary in the LU factorization. Thus, the *serial* complexity of this step is

$$\begin{aligned} C_{\text{step\_1}} &= C_{\text{LUP}}(\frac{n}{p},k,k) + (p-1)C_{\text{LUP}}(\frac{n}{p},2k,0) + ((p-1)2k+p)C_{\text{forw}}(\frac{n}{p},2k) \\ &= \left((16k^2+2k) - \frac{12k^2-3k}{p}\right)n - \frac{kp}{3}(56k^2+6k+1) + \frac{k}{6}(86k^2-27k+13) \text{ flops.} \end{aligned}$$

Notice, that we actually factor a lower triangular matrices if we imagine B_i being appended to A_i.

Each processor can work on its block row independently of each other. There is no interprocessor communication. Therefore, the parallel complexity of this step is

$$\begin{aligned} C_{\text{step\_1}}^{\text{par}} &= C_{\text{LUP}}(\frac{n}{p},2k,0) + (2k+1)C_{\text{forw}}(\frac{n}{p},2k) \\ &= 2k(8k+3)\frac{n}{p} - \frac{k}{3}(2k+1)(28k-5) \text{ flops.} \end{aligned}$$

Again, the work in this step is not completely balanced, as the A_i partially have different structures. Furthermore, processors 1 and p have only one off-diagonal block to compute.

2. Formation and solution of reduced system

For the sequel we split the matrices R_i, E_i, F_i in the following way:

$$R_i = \begin{bmatrix} \hat{R}_i \\ O \end{bmatrix}, \quad E_i = \begin{bmatrix} \hat{E}_{i1} \\ \hat{E}_{i2} \end{bmatrix}, \quad F_i = \begin{bmatrix} \hat{F}_{i1} \\ \hat{F}_{i2} \end{bmatrix}, \quad \mathbf{c}_i = \begin{bmatrix} \hat{\mathbf{c}}_{i1} \\ \hat{\mathbf{c}}_{i2} \end{bmatrix}. \tag{5.4}$$

As the A_i have maximal rank n_i, the diagonal elements of $\hat{R}_i$ do not vanish. The unknown $\boldsymbol{\xi}_i$ can now be determined by those rows of equation (5.3) which correspond to the zero rows of the R_i,

$$\begin{bmatrix} \hat{E}_{12} & & & \\ \hat{F}_{22} & \hat{E}_{22} & & \\ & \ddots & \ddots & \\ & & \hat{F}_{p-1,2} & \hat{E}_{p-1,2} \\ & & & \hat{F}_{p2} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_1 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{c}}_{12} \\ \vdots \\ \hat{\mathbf{c}}_{p2} \end{bmatrix} \tag{5.5}$$

The reduced system (5.5) has order $(p-1)m$. The system matrix S is blocked. All blocks are $m \times m$, except $\hat{E}_{12}$, which is $r \times m$, and $\hat{F}_{i2}$, which is $s \times m$. As solving $S\mathbf{z} = \mathbf{h}$ requires pivoting, we cannot apply cyclic reduction, but are restricted to Gaussian elimination. We proceed as in the other algorithms. The factorization is performed sequential by all processors involved in the computation. Each processors factors its portion of the reduced system and sends those data (rm

floating point numbers) to the next processor that that needs for factoring its portion of the system. The back-solving phase proceeds in the other direction, the message volume is only m.

With $k = r = s$ and $m = 2k$ the sequential complexity of solving (5.5) by Gaussian elimination becomes

$$C_{\text{step\_2}} = \frac{2k}{3}(13k^2 + 12k + 5)(p-1) - 2k^2(3k+2) \text{ flops.}$$

Because this step is performed sequentially, the parallel complexity is $C_{\text{step\_2}}$ plus the complexity of the communication,

$$C^{\text{par}}_{\text{step\_2}} = (p-1)\left(\frac{2k}{3}(13k^2 + 12k + 5) + 2\sigma + (2k^2 + 2k)\tau\right) - 2k^2(3k+2) \text{ flops.}$$

3. Back substitution

Knowing the vectors $\boldsymbol{\xi}_i$, $1 \le i < p$, the i-th processor can compute its section of $\mathbf{x}$ by

$$\begin{array}{rcll} \mathbf{x}_1 & = & \hat{R}_i^{-1}(\hat{\mathbf{c}}_{11} - \hat{E}_{11}\boldsymbol{\xi}_1), & \\ \mathbf{x}_i & = & \hat{R}_i^{-1}(\hat{\mathbf{c}}_{i1} - \hat{F}_{i1}\boldsymbol{\xi}_{i-1} - \hat{E}_{i1}\boldsymbol{\xi}_i), & 1 < i < p, \\ \mathbf{x}_p & = & \hat{R}_p^{-1}(\hat{\mathbf{c}}_{p1} - \hat{F}_{p1}\boldsymbol{\xi}_{p-1}). & \end{array} \tag{5.6}$$

Each processor can proceed independently, there is no interprocessor communication. Serial and parallel complexities of this step are

$$C_{\text{step\_3}} = \left(8k - 1 - \frac{4k}{p}\right)n - 2k(2k+1) \qquad \text{and} \qquad C^{\text{par}}_{\text{step\_3}} = (8k-1)\frac{n}{p},$$

respectively.

Redundancy and speedup

Again with the assumption that $k = m/2 = r = s$ and that $n_i = n/p$ the overall *serial* complexity of this algorithm is

$$C_{\text{dws}}(n,k,p) \approx \left(16 - \frac{12}{p}\right)k^2 n - 10pk^3 \text{ flops.} \tag{5.7}$$

Comparing (5.7) with the cost (2.2) of Gaussian elimination with pivoting, we obtain the redundancy

$$R_{\text{dws}}(n,k,p) = \frac{C_{\text{dws}}(n,k,p)}{C_{\text{GaussP}}(n,k)} \approx 4 - \frac{3}{p} - \frac{5}{2}\frac{pk}{n}, \tag{5.8}$$

which is, for large p, the same as with the two other algorithms.

The parallel complexity of the double width separator approach is

$$C^{\text{par}}_{\text{dws}} \approx 16k^2\frac{n}{p} + \left(\frac{26}{3}k^3 + 2\sigma + 2k^2\tau\right)p \text{ flops.} \tag{5.9}$$

Thus, speedup becomes

$$S_{\text{dws}}(n,k,p) = \frac{C_{\text{GaussP}}(n,k)}{C^{\text{par}}_{\text{dws}}(n,k,p)} \approx \frac{p}{4 + \left(\frac{13}{6}k + \frac{1}{2}\tau + \frac{\sigma}{2k^2}\right)\frac{p^2}{n}}, \tag{5.10}$$

yielding optimal processor number

$$p^{\text{opt}}_{\text{dws}} \approx \sqrt{\frac{24n}{13k}}\sqrt{\frac{1}{1 + \frac{3\tau}{13k} + \frac{3\sigma}{13k^3}}} \tag{5.11}$$

and corresponding optimal speedup

$$S^{\text{opt}}_{\text{dws}} \approx \sqrt{\frac{3n}{104k}}\sqrt{\frac{1}{1 + \frac{3\tau}{13k} + \frac{3\sigma}{13k^3}}}. \tag{5.12}$$

Memory requirements

Extra memory space for storing the $2kn$ entries of the block matrices F_i. The space needed for the local portions of the reduced system is about $12k^2$ floating point numbers.

Remarks

The incorporation of pivoting into parallel solvers for banded and in particular tridiagonal linear systems is not discussed often. Hegland [11] presents a variant of Wang's algorithm that allows pivoting for the solution of tridiagonal systems.

6 Discussion

The three approaches we discussed share the same algorithmic structure. There are two phases, factorization and back substitution, that are naturally parallelizable. The third phase, the solution of the reduced system, is completely sequential. This phase evidently is the bottleneck of these algorithms.

As the cost for the solution of the reduced system increases with the number of processors p, speedup will be maximal for some optimal number of processors. This means that the speedup increases only until a certain number p^{opt} of processors. p^{opt} depends on the size of the problem. The existence of p^{opt} means that these algorithms are not scalable. In Tab. 6.1 we list the speedup, the optimal number of processors and corresponding maximal speedups for the various algorithms.

algorithm	speedup $S(n,k,\sigma,\tau)$	p^{opt}	S^{opt}
d&c	$\frac{p}{4+\left(\frac{7}{3}k+\frac{\tau}{2}+\frac{\sigma}{k^2}\right)\frac{p^2}{n}}$	$\sqrt{\frac{12n}{7k}\frac{1}{1+\frac{3\tau}{14k}+\frac{3\sigma}{7k^3}}}$	$\frac{p^{\text{opt}}}{8}$
d&c with cr	$\frac{p}{4+\left(\frac{31}{3}k+4\tau+\frac{2\sigma}{k^2}\right)\frac{p\log_2 p}{n}}$	$\frac{12n}{31k}\frac{1}{1+\frac{12\tau}{31k}+\frac{6\sigma}{31k^3}}$	$\frac{p^{\text{opt}}}{4}\frac{1}{1+\log_2(p^{\text{opt}})}$
sws	$\frac{p}{4+\left(\frac{7}{3}k+\frac{\tau}{2}+\frac{\sigma}{k^2}\right)\frac{p^2}{n}}$	$\sqrt{\frac{12n}{7k}\frac{1}{1+\frac{3\tau}{14k}+\frac{3\sigma}{7k^3}}}$	$\frac{p^{\text{opt}}}{8}$
sws with cr	$\frac{p}{4+\left(\frac{7}{3}k+2\tau+\frac{2\sigma}{k^2}\right)\frac{p\log_2 p}{n}}$	$\frac{12n}{7k}\frac{1}{1+\frac{6\tau}{7k}+\frac{6\sigma}{7k^3}}$	$\frac{p^{\text{opt}}}{4}\frac{1}{1+\log_2(p^{\text{opt}})}$
symmetric sws with cr	$\frac{p}{4+\left(\frac{7}{3}k+\tau+\frac{2\sigma}{k^2}\right)\frac{p\log_2 p}{n}}$	$\frac{12n}{7k}\frac{1}{1+\frac{3\tau}{7k}+\frac{6\sigma}{7k^3}}$	$\frac{p^{\text{opt}}}{4}\frac{1}{1+\log_2(p^{\text{opt}})}$
dws	$\frac{p}{4+\left(\frac{13}{6}k+\frac{\tau}{2}+\frac{\sigma}{2k^2}\right)\frac{p^2}{n}}$	$\sqrt{\frac{24n}{13k}\frac{1}{1+\frac{3\tau}{13k}+\frac{3\sigma}{13k^3}}}$	$\frac{p^{\text{opt}}}{8}$

Table 6.1: Characteristic numbers for various algorithms

Clearly, the variants incorporating cyclic reduction are superior to the variants without. The denominator of the speedup formula of the latter containes a p^2 term in contrast to the $p\log_2(p)$ term for the former. By consequence, the optimal processor numbers of the algorithms with cyclic reduction are approximately the square of the optimal processor numbers of the algorithms without cyclic reduction. Single width separator (sws) and divide and conquer (d&c) approaches both without cyclic reduction behave equally. If the reduced system is solved by cyclic reduction (sws/cr), the single width separator approach is slightly faster than d&c/cr, the divide and conquer approach with cyclic reduction. The main advantage of sws and sws/cr are their ability to exploit the structure of symmetric problems.

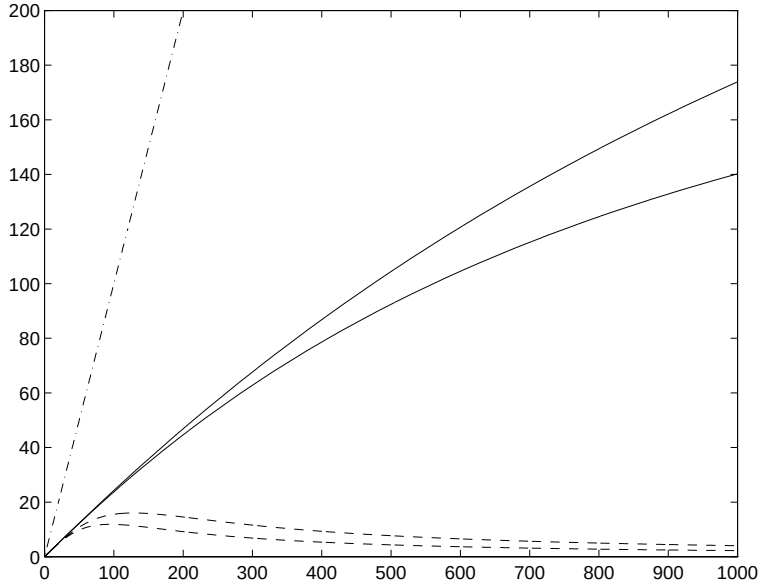


Figure 6.1: Theoretical speedups for $n = 100000$, $k = 10$, and $\tau = 1$. The pair of dashed lines corresponds to sws, the pair of solid lines to sws/cr. The upper and lower of the two lines correspond to $\sigma = 0$ and $\sigma = 1000$, respectively. The dash-dotted line indicates ideal speedup.

The *efficiency* of a parallel program is defined by

$$E(n,k,p) := \frac{S(n,k,p)}{p}. \tag{6.1}$$

Efficiency measures the fraction of the processor power that is actually utilized by the parallel program. It is evident from Table 6.1 that for fixed problem size the efficiency decreases if the processor number is increased. To have equal efficiency (isoefficiency [10]) when increasing the number of processors, the problem size has to be increased also. For these algorithms the processor number has to grow very rapidly to that end. From Table 6.1 and (6.1) we see that for the single width separator approaches the relations among n, k, and p are

$$p^2 = n \Big/ \left(\frac{7}{3}k + \frac{\tau}{2} + \frac{\sigma}{k^2} \right)$$

for sws and

$$p \log_2(p) = n \Big/ \left(\frac{7}{3}k + 2\tau + \frac{2\sigma}{k^2} \right)$$

for sws/cr. If communication were negligible, speedup and efficiency depended only on the ratio n/k. This the case only if k is large.

In Figure 6.1 the theoretical speedups of Table 6.1 are plotted for sws and sws/cr for the two values $\sigma = 1000$ and $\sigma = 0$ for a problem of size $n = 100000$ and $k = 10$. Here, we assumed that $\tau = 1$. $\sigma = 0$ stands for the case where the communication startup cost can be hidden behind computation. On the Paragon this can be done, at least partially, with asynchronous message passing primitives. The curves show first that speedups for sws are much smaller than for sws/cr. Furthermore, they show that the optimal processor number is relatively low for sws. For this problem size it is around 100. Speedup as well as efficiency are between 10% and 20%. With sws/cr much higher speedups are obtained ($p^{\text{opt}} = 15789$, $S^{\text{opt}} = 264$). However, the efficiency is not very high either. We remark here, that the speedup and efficiency would be satisfactory if they were given with respect to the performance of the *parallel* algorithm on one node as we would then neglect its high redundancy.

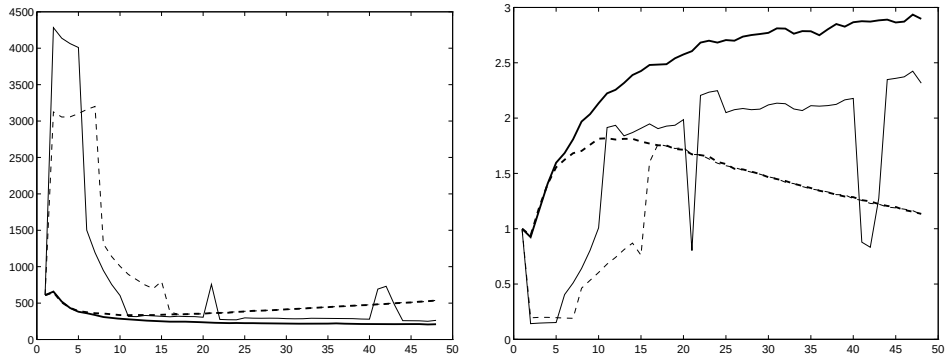


Figure 6.2: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 10000$ and $k = 10$. The thin/thick curves correspond to calculations with IEEE arithmetic turned on/off.

p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE	p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE
1	612	608	615	609	25	387	384	297	225
2	3125	648	4282	658	26	391	394	293	225
3	3055	511	4138	521	27	397	396	291	222
4	3057	433	4064	436	28	402	401	293	221
5	3103	391	4010	381	29	407	407	292	220
6	3158	375	1502	362	30	414	415	287	220
7	3200	367	1193	336	31	419	420	285	216
8	1315	357	952	309	32	428	425	286	217
9	1140	346	757	299	33	432	432	292	220
10	1006	335	604	285	34	440	438	294	218
11	894	335	318	273	35	447	444	288	218
12	821	337	314	270	36	451	453	288	221
13	750	336	331	262	37	456	458	288	217
14	699	336	325	255	38	465	465	286	213
15	799	340	319	251	39	465	471	281	215
16	380	344	312	245	40	478	473	279	212
17	345	347	319	245	41	484	483	692	211
18	348	347	316	245	42	495	488	731	212
19	353	353	314	239	43	497	497	477	211
20	351	355	306	236	44	508	504	259	210
21	364	364	758	233	45	512	508	258	212
22	364	365	276	227	46	517	520	256	212
23	373	368	272	225	47	523	527	251	207
24	382	379	271	227	48	538	536	263	210

Table 6.2: Times in milliseconds for sws and sws/cr for the smallest problem size $(n, k) = (10000, 10)$.

We ran the symmetric single width separator algorithm for 4 different problem sizes on the Intel Paragon at ETH Zurich [2]. This machine has 96 compute nodes based on Intel's i860XP RISC processor. The operating system version was OSF/1, Release 1.2.3, which exploits the message processors that complement the compute processor on each node. The message processors are necessary for asynchronous message passing. Each node has a memory of 32 MByte of which about 6 are occupied by the operating system. Larger problems can be solved by using the very slow secondary (disk) memory. For efficiency reasons storing on disk should be avoided. In our test we always used the local memory. If problems were too big to fit into this, we extrapolated timings from smaller problems which did fit into memory. (Extrapolated numbers are indicated by an asterisk, cf. Tab. 6.4 and Tab. 6.5.) In this way we avoid performance losses due to page swapping. On the other hand, speedups are smaller this way than actually observed as we neglect the overheads involved with solving large problems on a single processor.

One series of measurements (the numbers in one table) was always done with the same executable. The arrays were dimensioned according to the requirements of the problem running on the smallest number of processors. All times given are the best obtained in several measurements. The presented numbers contain only the time for the solution of the banded system. We assume that this algorithm is used in a program that computes the system matrix distributed over the processors. So, the time for constructing and distributing the matrix is not included in the shown times. A fortiori, the time to load the program from disk is not included. Loading the program may take longer than the computation.

In Tab. 6.2 the timings for the smallest problem size are presented. Figure 6.2(a) shows the corresponding plots. In Fig. 6.2(b) the speedups are given. For this small problem, only up to 48 nodes were used. The Fortran programs have been compiled with the IEEE flag turned on and off. It is observed in Fig. 6.2 as in the figures shown later that the cost of the floating point arithmetic according to the IEEE is very high. On the Intel Paragon, the IEEE division and square root are written in software. Also the exception handling for the multiplication is done in software. It can be observed in this and later figures, that the numbers obtained with IEEE arithmetic turned off behave much smoother and better according to theory than those obtained with IEEE arithmetic. These numbers sometimes behave in an erratic manner. Spikes seen in performance curves are *reproducible*.

The timings for the problem size $(n, k) = (100000, 10)$ are given in Tab. 6.3. The corresponding plots of times and speedups are in Fig. 6.3. Again, the curves of the timings obtained without IEEE arithmetic are much better than those with the IEEE flag turned on. The plot clearly indicates that the speedups without IEEE behave as theoretically predicted. The speedups obtained with the computations are slightly below the ones predicted. sws has its peak speedup at around $p = 35$. Speedup for sws/cr is increasing in the whole domain covered here. Efficiencies range from 15% to 20%. The timings for the runs with IEEE arithmetic show a sudden collapse at $p = 53$. The form of this performance (speedup) jump indicates that there are frequent cache misses until the local problem size gets so small that it fits into cache. The spike in the execution times for sws right before the jump ($p = 51, 52$), is not yet understood. It is presently under investigation by Intel engineers.

In Tab. 6.4 the timings for the problem size $(n, k) = (800000, 10)$ are listed with the corresponding plots in Fig. 6.4. In Table 6.4 the one processor times were obtained by linear extrapolation as neither of the problems fit into the memory of one processor. Linear extrapolation was motivated by the linear dependence of $C_{\text{Cholesky}}(n, k)$ in (2.4) with respect to n. The reference time of 46368 milliseconds is eight times the fastest one processor time in Tab. 6.3. The (800000,10) problem could be solved on one processor using the slow secondary memory in about 600 seconds! Speedups with respect to this number are greater than p. The speedups obtained with sws/cr are very satisfactory. Efficiency is close to 40%; however, with respect to the estimated one-processor time.

Table 6.5 gives the timings for the problem size $(n, k) = (800000, 40)$. Plots for timings and corresponding speedups are found in Fig. 6.5. Because of the wider band, speedups have decreased with respect to the example before. The one-processor time is 30 times the time measured for solving a banded system of order 33335 and half-bandwidth 40 on one node. For the non-IEEE version of the programs, in particular sws/cr, speedups are satisfactory. We observe speedups of

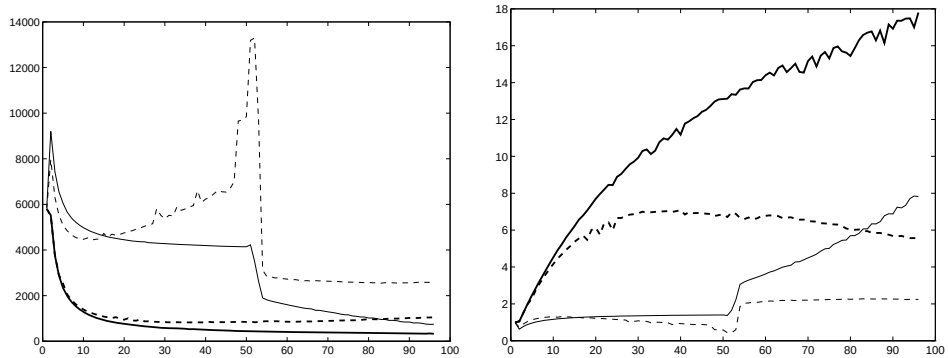


Figure 6.3: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 100000$ and $k = 10$. The thin/thick curves correspond to calculations with IEEE arithmetic turned on/off.

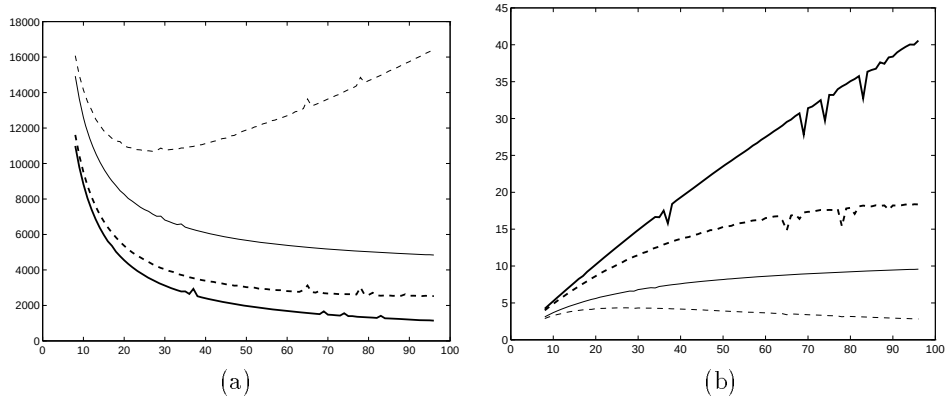


Figure 6.4: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 800000$ and $k = 10$. The thin/thick curves correspond to calculations with IEEE arithmetic turned on/off.

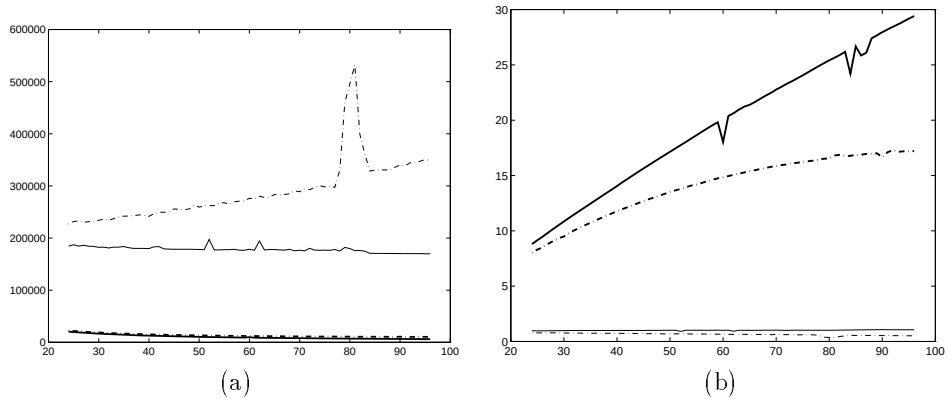


Figure 6.5: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 800000$ and $k = 40$. The thin/thick curves correspond to calculations with IEEE arithmetic turned on/off.

p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE	p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE
1	5805	5796	5858	5797	49	9704	851	4143	443
2	7966	5498	9200	5526	50	9845	846	4143	442
3	6314	3777	7442	3768	51	13194	864	4225	441
4	5533	2908	6550	2877	52	13294	833	3516	433
5	5086	2529	6052	2336	53	9623	842	2618	435
6	4849	2061	5669	1999	54	3116	870	1898	425
7	4671	1823	5421	1743	55	2843	879	1816	423
8	4561	1641	5236	1545	56	2830	867	1763	424
9	4488	1501	5098	1406	57	2776	880	1721	413
10	4468	1394	4980	1282	58	2784	873	1681	410
11	4532	1301	4886	1183	59	2742	858	1642	410
12	4434	1238	4792	1109	60	2726	856	1600	403
13	4473	1177	4740	1036	61	2686	852	1557	399
14	4483	1130	4672	983	62	2700	856	1530	403
15	4721	1080	4625	933	63	2672	858	1490	392
16	4561	1046	4572	884	64	2649	871	1453	388
17	4693	1015	4541	854	65	2655	868	1432	398
18	4642	1059	4508	821	66	2634	866	1414	392
19	4685	964	4476	785	67	2655	876	1356	386
20	4750	944	4447	753	68	2644	882	1352	398
21	4808	1007	4421	728	69	2638	882	1320	399
22	4846	908	4396	706	70	2633	893	1290	382
23	4919	901	4373	685	71	2624	895	1263	376
24	4966	968	4367	687	72	2618	897	1234	390
25	5043	871	4357	652	73	2610	909	1193	375
26	5091	870	4323	641	74	2600	911	1162	370
27	5155	860	4311	622	75	2591	913	1157	379
28	5807	848	4300	606	76	2590	918	1109	365
29	5522	844	4295	597	77	2584	920	1089	363
30	5363	839	4273	584	78	2578	931	1065	369
31	5523	828	4273	563	79	2564	952	1061	371
32	5497	834	4250	559	80	2577	961	1017	375
33	5906	832	4255	572	81	2562	963	1017	365
34	5741	829	4247	562	82	2563	961	995	356
35	5799	828	4235	538	83	2546	973	956	350
36	5883	826	4218	528	84	2565	975	955	347
37	5947	824	4221	532	85	2556	990	914	345
38	6604	829	4209	519	86	2560	991	915	356
39	6088	824	4189	505	87	2548	991	898	345
40	6231	818	4182	519	88	2558	1001	855	359
41	6315	847	4194	492	89	2559	1008	841	338
42	6461	832	4178	487	90	2563	1020	843	342
43	6609	837	4175	480	91	2564	1024	800	334
44	6553	837	4160	476	92	2582	1019	804	334
45	6538	837	4164	467	93	2589	1031	789	332
46	6694	841	4154	463	94	2582	1041	752	331
47	6994	843	4157	455	95	2581	1041	738	341
48	9651	857	4146	447	96	2585	1050	741	326

Table 6.3: Times in milliseconds for sws and sws/cr for the problem size $(n, k) = (100000, 10)$.

p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE	p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE
1	46368*	46368*	46368*	46368*	52	12060	3000	5605	1908
8	16078	11611	14926	10991	53	12111	2949	5572	1876
9	14971	10443	13645	9798	54	12229	2933	5542	1847
10	14153	9521	12597	8839	55	12294	2917	5517	1820
11	13497	8755	11798	8054	56	12404	2890	5488	1792
12	13024	8121	11129	7409	57	12448	2867	5454	1762
13	12502	7587	10593	6856	58	12539	2857	5434	1738
14	12172	7129	10100	6387	59	12610	2865	5410	1710
15	11848	6740	9712	5978	60	12694	2809	5386	1687
16	11639	6402	9323	5618	61	12776	2795	5360	1664
17	11422	6099	9019	5360	62	12921	2773	5333	1639
18	11273	5832	8754	5028	63	12961	2767	5310	1618
19	11233	5582	8468	4777	64	13100	2869	5295	1596
20	10987	5364	8269	4549	65	13638	3124	5275	1570
21	10910	5165	8033	4342	66	13242	2752	5253	1553
22	10875	4984	7876	4154	67	13330	2723	5227	1529
23	10784	4838	7705	3986	68	13424	2827	5216	1510
24	10766	4704	7533	3832	69	13530	2687	5198	1667
25	10714	4566	7402	3686	70	13636	2671	5181	1477
26	10719	4451	7305	3554	71	13730	2672	5167	1466
27	10686	4316	7146	3430	72	13838	2651	5147	1446
28	10733	4212	7025	3317	73	13915	2635	5127	1427
29	10869	4112	7033	3212	74	14046	2637	5106	1561
30	10753	4041	6814	3113	75	14159	2632	5097	1396
31	10777	3944	6728	3026	76	14245	2636	5082	1397
32	10802	3886	6635	2937	77	14376	2615	5065	1365
33	10804	3779	6560	2860	78	14856	3015	5046	1350
34	10874	3719	6590	2785	79	14555	2604	5036	1337
35	10875	3666	6412	2790	80	14671	2590	5027	1322
36	10935	3590	6343	2648	81	14764	2716	5011	1311
37	10964	3535	6273	2938	82	14871	2563	4991	1297
38	11035	3480	6208	2517	83	14974	2549	4990	1417
39	11057	3431	6157	2455	84	15084	2560	4972	1275
40	11128	3391	6102	2403	85	15212	2563	4964	1268
41	11186	3350	6036	2354	86	15306	2542	4945	1261
42	11253	3326	5995	2301	87	15410	2553	4939	1233
43	11300	3265	5936	2252	88	15527	2545	4925	1239
44	11397	3241	5904	2210	89	15643	2628	4922	1211
45	11433	3190	5855	2166	90	15744	2545	4900	1208
46	11553	3171	5812	2122	91	15873	2548	4895	1190
47	11583	3117	5776	2078	92	15973	2542	4885	1178
48	11696	3119	5745	2043	93	16080	2527	4873	1167
49	11810	3073	5704	2005	94	16210	2543	4857	1158
50	11889	3030	5674	1972	95	16311	2523	4857	1159
51	11949	3016	5634	1937	96	16418	2527	4839	1143

Table 6.4: Times in milliseconds for sws and sws/cr for the problem size $(n, k) = (800000, 10)$.

p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE	p	sws IEEE	sws no IEEE	sws/cr IEEE	sws/cr no IEEE
1	180000*	180000*	180000*	180000*	60	276228	12124	178552	9984
24	226844	22411	184503	20439	61	275458	12041	176584	8831
25	231692	21683	187147	19662	62	280148	11951	194330	8721
26	233003	21160	184480	18993	63	277826	11866	177127	8584
27	230277	20403	186144	18303	64	278210	11781	177955	8480
28	231328	19825	184409	17691	65	283594	11697	177978	8414
29	230978	19288	183907	17140	66	282490	11627	177274	8315
30	234083	18967	182190	16581	67	283695	11546	176810	8206
31	236343	18346	182589	16086	68	284913	11478	178497	8103
32	234138	17900	181003	15631	69	290630	11402	175562	8012
33	237303	17499	182652	15248	70	288933	11359	176733	7905
34	240290	17121	182476	14842	71	293822	11290	175385	7820
35	242065	16785	183876	14442	72	292764	11247	180244	7726
36	242193	16446	181550	14085	73	297453	11171	176878	7649
37	242828	16137	180058	13746	74	300384	11146	176583	7571
38	244528	15853	180032	13409	75	298085	11085	176912	7484
39	243552	15561	180281	13092	76	299822	11046	176364	7393
40	241679	15289	179901	12814	77	297281	11013	178067	7314
41	246965	15047	183271	12518	78	330815	10954	175096	7232
42	249553	14826	183510	12246	79	460348	10896	182069	7150
43	249418	14611	179230	11990	80	495903	10876	179871	7092
44	250911	14394	179165	11743	81	532276	10706	175859	7018
45	255827	14185	178057	11519	82	398907	10672	176166	6952
46	255093	13997	178723	11287	83	360599	10740	174826	6875
47	254598	13823	178592	11063	84	328344	10725	170723	7438
48	255689	13658	178415	10878	85	330594	10677	170576	6750
49	262859	13490	178375	10691	86	330469	10661	170541	6963
50	259417	13316	178172	10504	87	330785	10619	170399	6896
51	261530	13182	177579	10319	88	330404	10605	170413	6567
52	262061	13053	197348	10136	89	336000	10580	170273	6505
53	262293	12912	177257	9985	90	340024	10804	170334	6438
54	267280	12776	177247	9804	91	338540	10520	170168	6381
55	267727	12679	177687	9653	92	344502	10414	170226	6331
56	265153	12556	177645	9505	93	344514	10501	170017	6274
57	269418	12425	178342	9349	94	347304	10474	170139	6224
58	270875	12311	176771	9211	95	350723	10467	169940	6168
59	271509	12210	176606	9080	96	348364	10458	169933	6121

Table 6.5: Times in milliseconds for sws and sws/cr for the problem size $(n, k) = (800000, 40)$.

almost 30 with 96 processors. As with the half-bandwidth 10, the speedup curve for sws/cr is still increasing at the higher end. The IEEE versions are *very* slow. Ratios of around 30 are found. The ratio was not as large in the $(800000, 10)$ problem. Thus, the loss is in the solution of the reduced system. This part of the program is written in plain Fortran. There are in contrast to the other parts of the program which handle the banded (A_i) and blocked (B_i, E_i) matrices no calls to LAPACK routines.

7 Conclusions

In conclusion, one can say that direct methods for solving banded systems of equations are a reasonable solution path on parallel machines if the bandwidth of the matrix is *very* narrow. If possible the reduced system must be solved with cyclic reduction.

These algorithms are not scalable. There will always be a processor number p^{opt} for which speedup is highest. This number depends first of all on the ratio n/k, cf. Tab. 6.1. The speedup curve is very flat at its peak, cf. Fig. 6.2. As efficiency decreases with the processor number, it is not worth to actually solve a problem on a processor number close to p^{opt}. However, in our largest numerical experiments we did not get even close to the peak speedup with 96 processors.

Speedups and efficiencies of the investigated algorithms will be low as their redundancy, i.e. the *algorithmic* overhead by parallelizing Gaussian elimination is high. However, as seen in the large numerical examples, it is difficult to compare with a one-processor solution, as it does not even exist. It may be impossible (or excessively slow) to solve a large problem on a small number of processors because of its size.

The structure of the algorithms for solving sparse systems of linear equations that are parallelized by domain decomposition techniques is the same as the one of the investigated algorithms. If the underlying domain is long and narrow and if one-way dissection is used to define the subdomains, a matrix is obtained with at least locally narrow bandwidth. One of the above algorithms can be used to solve such a linear system. Only the band factorization has to be replaced by a sparse solver. As the latter is more expensive than a simple band solver, the dominance of the cost of the solution of the reduced system is reduced and speedups should be higher than the ones reported here.

Dongarra and Sameh [7] propose to solve the reduced system iteratively. If the number of iterations does not depend on the number of processors used, the algorithm becomes scalable. This assumption appears to be questionable. Nevertheless, it seems that the only way to *scalably* solve banded systems on MPP computers is by means of some iterative procedure.

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. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[2] P. Arbenz. First experiences with the Intel Paragon. *SPEEDUP*, 8(2), 1994.

[3] S. Bondeli. Divide and conquer: A parallel algorithm for the solution of a tridiagonal linear system of equations. *Parallel Computing*, 17:419–434, 1991.

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

[5] J. J. Dongarra. Performance of various computers using standard linear equation software equations software. Tech. Report CS-89-85, University of Tennessee, Computer Science Department, Knoxville, TN, August 1994.

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

[7] J. J. Dongarra and A. H. Sameh. On some parallel banded system solvers. *Parallel Computing*, 1:223–235, 1984.

[8] J. A. George. Numerical experiments with dissection methods to solve n by n grid problems. *SIAM J. Numer. Anal.*, 14:345–363, 1973.

[9] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 2nd edition, 1989.

[10] A. Y. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architecture. *IEEE Parallel & Distributed Technology*, 1:12–21, August 1993.

[11] M. Hegland. On the parallel solution of tridiagonal systems by wrap-around partitioning and incomplete LU factorizattion. *Numer. Math.*, 59:453–472, 1991.

[12] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, New York, 1993.

[13] S. L. Johnsson. Solving narrow banded systems on ensemble architectures. *ACM Trans. Math. Softw.*, 11:271–288, 1985.

[14] D. Lawrie and A. Sameh. The computation and communication complexity of parallel banded system solves. *ACM Trans. Math. Softw.*, 10:185–195, 1984.

[15] V. Mehrmann. Divide and conquer methods for block tridiagonal linear systems. *Parallel Computing*, 19:257–279, 1993.

[16] U. Meier. A parallel partition method for solving banded systems of linear equations. *Parallel Computing*, 2:33–45, 1985.

[17] S. J. Wright. Parallel algorithms for banded linear systems. *SIAM J. Sci. Stat. Comput.*, 12:824–842, 1991.