Report

A survey of direct parallel algorithms for banded linear systems

Author(s):
Arbenz, Peter; Gander, Walter

Publication Date:
1994

Permanent Link:
https://doi.org/10.3929/ethz-a-001382251

Rights / License:
In Copyright - Non-Commercial Use Permitted
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
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.

© 1994 Departement Informatik, ETH Zürich
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 satisfactorily only if the ratio of bandwidth and matrix order is very small. As a result of the high redundancy of the parallel algorithms efficiencies 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

\[ Ax = b \]  \hspace{1cm} (1.1)

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

\[ a_{ij} = 0 \quad \text{for } i - j > r \text{ and } j - i > s. \]  \hspace{1cm} (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.
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 $x$ and $b$ are partitioned in the form

$$
\begin{pmatrix}
A_1 & B_1 \\
C_2 & A_3 & B_3 \\
\vdots & \ddots & \ddots \\
& \ddots & \ddots \\
& & C_p & A_p
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{p-1} \\
x_p
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_{p-1} \\
b_p
\end{pmatrix},
$$

(3.1)
where $A_i \in \mathbb{R}^{n_i \times n_i}, x_i, b_i \in \mathbb{R}^{n_i}, \sum_{i=1}^{p} n_i = n$. 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.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{matrix_structure.png}
\caption{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.}
\end{figure}

We assume that we have $p$ processors available the $i$th of which deals with the $i$th block row in (3.1); the matrices $A_i, B_i, C_i$, and the vectors $x_i$ and $b_i$ reside in the local memory of the $i$th processor. Of the matrices $B_i \in \mathbb{R}^{n_i \times n_i + s}, i < p$, and $C_i \in \mathbb{R}^{n_i \times n_i}, i > 1$, 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 $C_i = [\hat{C}_i \hat{B}_i]$. 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 $F_i$,

$$E_i = U_i^{-1} L_i^{-1} \hat{B}_i, \quad F_i = U_i^{-1} L_i^{-1} \hat{C}_i,$$

and $e_i = U_i^{-1} L_i^{-1} b_i$ 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

$$C_{\text{step1}} = pC_{\text{mv}}(\frac{n}{p}, k) + (p-1)\frac{k(k+1)}{2} + ((p-1)k + p)C_{\text{ssav}}(\frac{n}{p}, k) + ((p-1)k + p)C_{\text{scal}}(\frac{n}{p}, k)$$

$$= \left(8k^2 + 3k - 1 - \frac{k}{p}(6k - 2)\right) n - \frac{kp}{2}(k + 1)(8k + 5) + \frac{k}{3}(k + 1)(8k + 1) \text{ flops.}$$

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

   $$\begin{pmatrix}
   I_{n_1} & E_1 \\
   F_2 & I_{n_2} & E_2 \\
   \vdots & \ddots & \ddots \\
   \vdots & \ddots & E_{p-1} \\
   F_p & \cdots & \cdots & I_{n_p}
   \end{pmatrix}
   \begin{pmatrix}
   x_1 \\
   x_2 \\
   \vdots \\
   x_{p-1} \\
   x_p
   \end{pmatrix}
   = \begin{pmatrix}
   c_1 \\
   c_2 \\
   \vdots \\
   c_{p-1} \\
   c_p
   \end{pmatrix},
   \tag{3.2}
   $$

   where $E_i = A_i^{-1} B_i$, $F_i = A_i^{-1} C_i$, and $e_i = A_i^{-1} b_i$. 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 $F_i$,

   $$E_i = U_i^{-1} L_i^{-1} \hat{B}_i, \quad F_i = U_i^{-1} L_i^{-1} \hat{C}_i,$$

   and $e_i = U_i^{-1} L_i^{-1} b_i$ 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

   $$C_{\text{step1}} = pC_{\text{mv}}(\frac{n}{p}, k) + (p-1)\frac{k(k+1)}{2} + ((p-1)k + p)C_{\text{ssav}}(\frac{n}{p}, k) + ((p-1)k + p)C_{\text{scal}}(\frac{n}{p}, k)$$

   $$= \left(8k^2 + 3k - 1 - \frac{k}{p}(6k - 2)\right) n - \frac{kp}{2}(k + 1)(8k + 5) + \frac{k}{3}(k + 1)(8k + 1) \text{ flops.}$$

5
The term \( k(k^2 - 1)/3 \) stems from the computation of \( L_i^{-1} \tilde{B}_i \), which preserves the sparsity structure of \( \tilde{B}_1 \).

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

\[
C_{\text{step}}^{\text{par}} = C_{\text{LU}}\left(\frac{n}{p}k\right) + \frac{k}{3}(k^2 - 1) + (k + 1)C_{\text{forw}}\left(\frac{n}{p}k\right) + (2k + 1)C_{\text{back}}\left(\frac{n}{p}k\right)
\]

\[
= (8k^2 + 3k - 1)\frac{n}{p} - \frac{1}{2}k(k + 1)(8k + 5) \text{ flops.}
\]

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 \( x_i, 1 \leq i < p \), and the first \( s \) components of the \( x_i, 2 \leq i \leq p \), respectively. More precisely, let \( \tilde{E}_i, \tilde{F}_i, x_i, \) and \( c_i \) be partitioned into their first \( s \), middle \( n_i - r - s \), and last \( r \) rows,

\[
\tilde{E}_i = \begin{bmatrix} \tilde{E}_{i1} \\ \tilde{E}_{i2} \\ \tilde{E}_{i3} \end{bmatrix}, \quad \tilde{F}_i = \begin{bmatrix} \tilde{F}_{i1} \\ \tilde{F}_{i2} \\ \tilde{F}_{i3} \end{bmatrix}, \quad x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix}, \quad c_i = \begin{bmatrix} c_{i1} \\ c_{i2} \\ c_{i3} \end{bmatrix}.
\]

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

\[
\begin{bmatrix}
I_r & \tilde{E}_{13} \\
\tilde{F}_{21} & I_s \\
\tilde{F}_{23} & O_{r \times s} & I_r \\
\vdots & \ddots & \ddots & \ddots \\
\tilde{F}_{p-1,3} & O_{r \times s} & I_r & \tilde{F}_{p-1,3} \\
\tilde{F}_{p,3} & I_s 
\end{bmatrix}
\begin{bmatrix}
x_{13} \\
x_{s1} \\
x_{s2} \\
x_{s3} \\
x_{p-1,1} \\
x_{p-1,3} \\
x_{p,1} 
\end{bmatrix}
= \begin{bmatrix}
e_{13} \\
e_{s1} \\
e_{s2} \\
e_{s3} \\
e_{p-1,1} \\
e_{p-1,3} \\
e_{p,1} 
\end{bmatrix}
\]

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 \( Sz = h \). 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 \( (2r + s) \times ((2r + s) \times s) \) 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 \( \tilde{E}_{i3} \) and \( c_{i3} \); in the backsubstitution phase the messages that are sent by processor \( i \) to processor \( i-1 \) have length \( s \). They comprise \( x_{i+1,1} \).
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
\[
\sigma + n\tau.
\]
\( \sigma \) 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. \( \tau \) 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,
\[
C_{\text{par,step,2}} = (p - 1) \left( \frac{k}{6} (28k^2 + 45k - 1) + 2\sigma + (k^2 + 2k)\tau \right) - k(4k^2 + 4k - 1) \text{ flops.}
\]

For an Intel Paragon running the operating system OSF/1 Release 1.2 the startup time for message passing is about 65\( \mu \)sec. 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
\[
\begin{pmatrix}
T_1 & U_1 & U_2 \\
V_2 & T_2 & U_3 \\
& & \ddots \\
& & & T_{p-2} & U_{p-1} \\
& & & & V_{p-1} & T_{p-1}
\end{pmatrix}
\begin{pmatrix}
\zeta_1 \\
\zeta_2 \\
\vdots \\
\zeta_{p-2} \\
\zeta_{p-1}
\end{pmatrix} =
\begin{pmatrix}
\eta_1 \\
\eta_2 \\
\vdots \\
\eta_{p-2} \\
\eta_{p-1}
\end{pmatrix},
\]

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

\[
\begin{bmatrix}
T_1 & U_1 & \zeta_1 \\
T_3 & U_3 & \zeta_3 \\
\vdots & \ddots & \vdots \\
T_{p_1-2} & U_{p_1-2} & \zeta_{p_1-2} \\
V_2 & U_2 & \zeta_2 \\
V_4 & U_4 & \zeta_4 \\
\vdots & \ddots & \vdots \\
V_{p_2} & U_{p_2} & \zeta_{p_2}
\end{bmatrix} = \begin{bmatrix}
\eta_1 \\
\eta_3 \\
\vdots \\
\eta_{p_1-2} \\
\eta_2 \\
\eta_4 \\
\vdots \\
\eta_{p_2}
\end{bmatrix}
\]

(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,

\[
\begin{bmatrix}
I & I \\
I & I \\
\vdots & \ddots & \vdots \\
I & I \\
T_2 & U_2 \\
\hat{V}_4 & \hat{T}_4 \\
\vdots & \ddots & \vdots \\
\hat{V}_{p_2} & \hat{T}_{p_2}
\end{bmatrix}
\begin{bmatrix}
T_{3}^{-1}U_1 & \zeta_1 \\
T_{3}^{-1}V_3 & \zeta_3 \\
\vdots & \vdots \\
T_{p_1-2}^{-1}U_{p_1-2} & \zeta_{p_1-2} \\
T_{p_1}^{-1}V_{p_1} & \zeta_{p_1}
\end{bmatrix}
= \begin{bmatrix}
\eta_1 \\
\eta_3 \\
\vdots \\
\eta_{p_1-2} \\
\eta_2 \\
\eta_4 \\
\vdots \\
\eta_{p_2}
\end{bmatrix}
\]

(3.7)

where

\[
\hat{T}_k = T_k - U_kT_{k-1}v_{k-1} - V_kT_{k+1}^{-1}u_{k+1},
\]

\[
\hat{V}_k = -V_kT_{k+1}^{-1}v_{k+1}, \quad \hat{U}_k = -U_kT_{k+1}^{-1}u_{k+1},
\]

\[
\hat{\eta}_k = \eta_k - V_kT_{k-1}^{-1}\eta_{k-1} - U_kT_{k+1}^{-1}\eta_{k+1}.
\]

The \( T_k, U_k, V_k \), and \( \eta_k \) are stored in the memory of processor \( k \). Therefore, the odd-numbered processors can compute \( T_{k-1}^{-1}U_k, T_{k}^{-1}V_k \), 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

\[
\begin{bmatrix}
\hat{T}_2 & \hat{U}_2 \\
\hat{V}_4 & \hat{T}_4 \\
\vdots & \ddots & \ddots \\
\hat{V}_{p_2} & \hat{U}_{p_2} & \hat{T}_{p_2}
\end{bmatrix}
\begin{bmatrix}
\zeta_2 \\
\zeta_4 \\
\vdots \\
\zeta_{p_2}
\end{bmatrix}
= \begin{bmatrix}
\hat{\eta}_2 \\
\hat{\eta}_4 \\
\vdots \\
\hat{\eta}_{p_2}
\end{bmatrix}
\]

(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 $z$ are obtained by back-substitution.

Interestingly, the nonzero structures of $T_k, U_k$, and $V_k$ 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

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

and

$$C_{\text{par,step,2,cr}} \propto \log_2 |p - 1| \left( \frac{k}{6} (124k^2 + 69k - 7) + 4\sigma + (4k^2 + 4k) \tau \right) + \frac{1}{6} k(16k^2 + 33k - 7) \text{ flops},$$

respectively.

3. Back substitution

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

$$\begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix} = \begin{bmatrix} c_{11} \\ c_{12} \end{bmatrix} - \begin{bmatrix} E_{11} \\ E_{12} \end{bmatrix} x_{21},$$

$$x_{i2} = c_{2} - \hat{F}_{i2} x_{i-1,3} - \hat{E}_{i2} x_{i+1,1}, \quad 1 < i < p,$$

(3.9)

$$\begin{bmatrix} x_{p2} \\ x_{p3} \end{bmatrix} = \begin{bmatrix} c_{p2} \\ c_{p3} \end{bmatrix} - \begin{bmatrix} F_{p2} \\ F_{p3} \end{bmatrix} x_{p-1,3}.$$

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

$$C_{\text{step,3}} = 4k \left( 1 - \frac{1}{p} \right) n - 8k^2 p + 12k^2 \text{ flops} \quad \text{and} \quad C_{\text{step,3}}^{\text{par}} = 4k n \frac{n}{p} - 8k^2 \text{ flops},$$

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

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

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 $C_{\text{step,2}}, C_{\text{step,3}}$, and $C_{\text{Gauss}}$. Thus,

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

Notice, that $\frac{p k}{n} \leq 1$. Redundancy measures the overhead work that is introduced by parallelizing an algorithm. In this algorithm it is caused by the computation of the matrices $E_i$ and $F_i$ 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

$$C_{\text{d&c}}^{\text{par}} = C_{\text{step,1}}^{\text{par}} + C_{\text{step,2}}^{\text{par}} + C_{\text{step,3}}^{\text{par}} \approx 8k^2 n \frac{n}{p} + \left( \frac{14}{3} k^3 + 2\sigma + k^2 \tau \right) p \text{ flops,} \quad (3.12)$$

9
Here, optimal processor number and optimal speedup are 

\[ p_{\text{opt}}^{\text{d&c}} \approx \frac{12n}{7k} \sqrt{\frac{1}{1 + \frac{37}{12k} + \frac{3\sigma}{nk}}} \]  

(3.14) 

and yields an optimal speedup of

\[ S_{\text{opt}}^{\text{d&c}} := S(n, k, p_{\text{opt}}) \approx \sqrt{\frac{3n}{112k}} \sqrt{\frac{1}{1 + \frac{37}{12k} + \frac{3\sigma}{nk}}} = \frac{1}{8} p_{\text{opt}}^{\text{d&c}}. \]  

(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

\[ C_{\text{d&c,cr}}(n, k, p) = C_{\text{step,1}} + C_{\text{step,2,cr}} + C_{\text{step,3}} \approx \left( 8k^2 - 6k^2 \frac{1}{p} \right) n + \frac{50}{3} pk^3 \text{flops} \]  

(3.16)

whereas the parallel complexity is

\[ C_{\text{d&c,cr}}^{\text{par}} = C_{\text{step,1}}^{\text{par}} + C_{\text{step,2,cr}}^{\text{par}} + C_{\text{step,3}}^{\text{par}} \approx 8k^2 \frac{n}{p} + \left( \frac{62}{3} k^3 + 4\sigma + 8k^2 \tau \right) \log_2(p) \text{flops}. \]  

(3.17)

From this we get the redundancy

\[ R_{\text{d&c,cr}}(n, k, p) := \frac{C_{\text{d&c,cr}}(n, k, p)}{C_{\text{Gauss}}(n, k)} \approx 4 - \frac{3}{p} + \frac{25}{3} \frac{pk}{n}. \]  

(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

\[ S_{\text{d&c,cr}}(n, k, p) := \frac{C_{\text{Gauss}}(n, k)}{C_{\text{d&c,cr}}(n, k, p)} \approx \frac{p}{4 + \left( \frac{31}{3k} k + 4\tau + \frac{2\sigma}{k^2} \right) \log_2(p)}. \]  

(3.19)

Here, optimal processor number and optimal speedup are

\[ p_{\text{opt}}^{\text{d&c,cr}} \approx \frac{12n}{31k} \sqrt{\frac{1}{1 + \frac{12\tau}{31k} + \frac{6\sigma}{31k}}} \]  

(3.20) 

and

\[ S_{\text{opt}}^{\text{d&c,cr}} \approx \frac{3n}{31k + 12\tau + 6\sigma/k^2} \frac{1}{1 + \log_2 \left( \frac{12n}{31k + 12\tau + 6\sigma/k^2} \right)} \frac{1}{4 p_{\text{opt}}^{\text{d&c,cr}} + 1 + \log_2(1/p_{\text{opt}}^{\text{d&c,cr}})}. \]  

(3.21)

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

The matrices $\tilde{E}_i$ and $\tilde{F}_i$ 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 $k^2n - \frac{1}{2}kp^3$ 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 $x$ and $b$ are partitioned in the form

$$
\begin{pmatrix}
A_1 & B_1 & C_1 \\
B_1 & D_1 & C_2 \\
C_2 & B_2 & A_2 \\
\vdots & \ddots & \ddots \\
C_{2p-3} & D_{2p-2} & C_{2p-2} \\
B_{2p-2} & A_{p} & C_{2p-1}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
\xi_1 \\
x_2 \\
\vdots \\
\xi_{p-1} \\
x_p
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
\beta_1 \\
b_2 \\
\vdots \\
\beta_{p-1} \\
b_p
\end{pmatrix},
$$

where $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i \in \mathbb{R}^{n_i \times k}$, $C_i \in \mathbb{R}^{k \times n_i}$, $D_i \in \mathbb{R}^{k \times k}$, $x_i, b_i \in \mathbb{R}^{n_i}$, $\xi_i, \beta_i \in \mathbb{R}^k$, $k = \max(r, s)$, 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 $b_i$ and $\xi_i$. (Some of the variables may not be present on processor 1 and $p$.)
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

\[
\begin{bmatrix}
A_1 & B_1 \\
A_2 & B_2 & B_3 \\
\vdots & \ddots & \ddots & \ddots \\
A_{p-1} & \cdots & B_{2p-3} & B_{2p-1} \\
C_1 & C_2 & \cdots & \cdots & D_1 \\
C_3 & \ddots & \iddots & \ddots & \vdots \\
\vdots & \ddots & \iddots & \ddots & \ddots \\
C_{2p-3} & C_{2p-1} & \cdots & \cdots & D_{p-1}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{p-1} \\
x_p
\end{bmatrix} =
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_{p-1} \\
b_p
\end{bmatrix},
\]

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. \( \square \)

**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

\[
\begin{pmatrix}
R_1 & R_2 & \cdots & R_{p-1} \\
& E_1 & E_2 & \cdots & E_{2p-3} & E_{2p-2} \\
& T_1 & U_1 & & & \\
V_2 & T_2 & \cdots & & & \\
& & \cdots & & & \\
V_{p-1} & & & & & U_{p-2}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{p-1} \\
x_p \\
\xi_1 \\
\xi_2 \\
\vdots \\
\xi_{p-1}
\end{pmatrix}
= \begin{pmatrix}
e_1 \\
e_2 \\
\vdots \\
e_{p-1} \\
e_p \\
\gamma_1 \\
\gamma_2 \\
\vdots \\
\gamma_{p-1}
\end{pmatrix}
\] (4.3)

where

\[
E_{2i-2} = L_{i-1}^{-1} B_{2i-2}, \quad E_{2i-1} = L_{i-1}^{-1} B_{2i-1}, \quad F_{2i-2} = C_{2i-2} R_{i-1}^{-1}, \quad F_{2i-1} = C_{2i-1} R_{i-1}^{-1},
\]

\[
e_i = L_i^{-1} b_i, \quad \gamma_i = \beta_i - F_{2i-1} e_i - F_{2i} e_{i+1},
\]

\[
T_i = D_{i-1} - F_{2i-1} E_{2i-1} - F_{2i} E_{2i}, \quad U_i = -F_{2i} E_{2i+1}, \quad V_i = -F_{2i-1} E_{2i-2}.
\] (4.4)

Figure 4.2: Non-zero structure of the (a) \( L \) and (b) \( R \) factor of the LU decomposition of \( \hat{A} \). 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 \( e_i \). Furthermore, each processor computes its portion of the matrix and right hand side of the reduced system (4.6),

\[
\begin{bmatrix}
-F_{2i-2} E_{2i-2} & -F_{2i-1} E_{2i-1} \\
-F_{2i-1} E_{2i-2} & D_i - F_{2i-1} E_{2i-1}
\end{bmatrix} \in \mathbb{R}^{3k \times 2k}
\quad \text{and} \quad
\begin{bmatrix}
-F_{2i-2} e_i \\
\beta_i - F_{2i-1} e_{i+1}
\end{bmatrix} \in \mathbb{R}^{2k},
\] (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

\[
C_{\text{step,}\hat{A}} = pC_{LU}(n/p - k, k) + 2(p-1) k(k^2 - 1)/3 + ((p-1)k + p) C_{\text{forward}}(n/p - k, k)
\]

\[
+ (p-1)kC_{\text{backward}}(n/p - k, k) + pk(2n/p - k + 1) + (p-1)2k \left(\frac{k^2}{3} + \frac{k}{2} + \frac{1}{6}\right) + (p-2)2k^2(k+1)
\]

\[
= \left(8k^2 + 4k\right) n - k(p-1)(6k^2 + 9k + 1) + k(8k^2 - 12k + 1) \text{ flops}.
\]
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

\[ C_{\text{par,1}}^{\text{step}} = C_{LU}(n/k) + 2(k^2 + 1)/3 + (k + 1)C_{\text{torr}}(n/k) \]
\[ + kC_{\text{back}}(n/k) + k(2n/k + 1) + 2k \left( \frac{1}{3}k + \frac{1}{2} k + \frac{1}{6} \right) + 2k^2(k + 1) \]
\[ = (8k^2 + 4k)n/k - k \left( 16k^3 + 9k + 1 \right) \text{ flops}. \]

2. Formation and solution of reduced system

The reduced system is

\[ S\xi = \begin{pmatrix} T_1 & U_1 & \cdots & \cdots & U_{p/2-1} \\ V_1 & T_2 & \cdots & \cdots & V_{p/2-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \cdots & \cdots & \ddots & V_{p-2} & T_{p-1} \\ V_{p-1} & \cdots & \cdots & V_{p-2} & T_{p-1} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_{p-2} \\ \xi_{p-1} \end{pmatrix} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{p-2} \\ \gamma_{p-1} \end{pmatrix}. \tag{4.6} \]

The matrix \( S \) is the Schur complement of \( \bigoplus_{i=1}^n A_i \) in \( \hat{A} \). 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 \( \gamma_i \) are stored on processor \( i - 1 \) at the beginning of the solution process.

Block Gaussian elimination of (4.6) costs

\[ C_{\text{step,2}} = k \left( 28k^2 + 27k - 7 \right)(p - 1) - k^3(4k + 3) \text{ flops}. \]

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

\[ C_{\text{par,2}}^{\text{step}} = (p - 1) \left( k \left( 28k^2 + 27k - 7 \right) + 2\sigma + k(k + 2)(k - 7) \right) \text{ flops}. \]

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

\[ C_{\text{step,2,cr}} \leq (p - 1)k \left( 76k^2 + 21k - 7 \right) \text{ flops} \]

and

\[ C_{\text{par,2,cr}}^{\text{step}} \leq \log_2 |p - 1| \left( k \left( 76k^2 + 21k - 7 \right) + 4\sigma + (4k^2 + 4k)(k - 7) \right) + \frac{1}{6}k(4k^3 + 9k - 7) \text{ flops}, \]

respectively.

3. Back substitution

Knowing the vectors \( \xi_i, \ 1 \leq i < p \), the \( i \)-th processor can compute its section of \( \mathbf{x} \) by

\[ \mathbf{x}_i = R_i^{-1}(c_i - E_i\xi_i), \quad \mathbf{x}_1 = R_1^{-1}(c_1 - E_1\xi_1), \]
\[ \mathbf{x}_i = R_i^{-1}(c_i - E_{2i-2}\xi_{i-1} - E_{2i-1}\xi_i), \quad 1 < i < p, \]
\[ \mathbf{x}_p = R_p^{-1}(c_p - E_{2p-2}\xi_{p-1}). \tag{4.7} \]

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

\[ C_{\text{step,3}} = pC_{\text{back}}(n/k) + 4(p - 1)k(n/k) = \left( 6k - 1 - \frac{4k}{p} \right)n - 7k^2p + 4k^2, \]

14
and
\[ C_{\text{step}3}^{\text{par}} = (6k - 1) \frac{n}{p} - 7k^2, \]
respectively.

**Redundancy and speedup**

We proceed as in the previous section. We assume that \( k \coloneqq r = s \ll n \). 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

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

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

\[ R_{\text{sws}}(n, k, p) := \frac{C_{\text{sws}}(n, k, p)}{C_{\text{Gauss}}(n, k)} \approx 4 - \frac{3}{p} - \frac{5 pk}{3 n}, \]

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),

\[ C_{\text{sws}}^{\text{par}} \approx 8k^2 \frac{n}{p} + \left( \frac{14}{3} k^3 + 2\sigma + k^2 \tau \right) p \text{ flops.} \quad (4.10) \]

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

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

whereas the parallel complexity is

\[ C_{\text{sws,cr}}^{\text{par}} \approx 8k^2 \frac{n}{p} + \left( \frac{14}{3} k^3 + 4\sigma + 4k^2 \tau \right) \log_2(p) \text{ flops.} \quad (4.12) \]

From this we get the redundancy

\[ R_{\text{sws,cr}}(n, k, p) := \frac{C_{\text{sws,cr}}(n, k, p)}{C_{\text{Gauss}}(n, k)} \approx 4 - \frac{3}{p} - \frac{7 pk}{3 n}. \quad (4.13) \]

The speedup is given by

\[ S_{\text{sws,cr}}(n, k, p) := \frac{C_{\text{Gauss}}(n, k)}{C_{\text{sws,cr}}(n, k, p)} \approx \frac{p}{4 + \left( \frac{14}{3} k + 2\tau + 2\sigma \frac{1}{k^2} \right) \log_2(p)} \cdot \frac{p \log_2(p)}{n}. \quad (4.14) \]

Here, optimal processor number and optimal speedup are

\[ p_{\text{opt}}^{\text{sws,cr}} \approx \frac{12n}{7k} \frac{1}{1 + \frac{4\sigma}{k^2} + \frac{6\tau}{k}}, \quad (4.15) \]

and

\[ S_{\text{opt}}^{\text{sws,cr}} \approx \frac{3n}{1k + 6\tau + 6\sigma/k^2} \frac{1}{1 + \log_2 \left( \frac{12n}{7k} \frac{1}{1 + \frac{4\sigma}{k^2} + \frac{6\tau}{k}} \right)} = \frac{1}{4} \frac{p_{\text{opt}}^{\text{sws,cr}}}{1 + \log_2(p_{\text{opt}}^{\text{sws,cr}})}, \quad (4.16) \]

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 $\sigma$. 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 $x$ and $b$ are partitioned in the form (cf. Fig. 5.1a)

\[
\begin{pmatrix}
A_1 & B_1 & \\
C_2 & A_2 & B_2 \\
& \ddots & \ddots \\
& & C_{p-1} & A_{p-1} & B_{p-1} \\
& & & C_p & A_p \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{p-1} \\
x_p \\
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_{p-1} \\
b_p \\
\end{pmatrix}
\]

(5.1)

where $x_i \in \mathbb{R}^{n_i}$, $\xi_i \in \mathbb{R}^{m}$, and $b_1 \in \mathbb{R}^{n_1}$, $b_p \in \mathbb{R}^{n_p}$, $b_i \in \mathbb{R}^{n_i + m}$, $1 < i < p$. The $A_i$ are rectangular! Here, we have $\sum_{i=1}^{p} n_i + (p - 1)m = n$. 

16
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),

\[
\begin{pmatrix}
A_1 & B_1 & C_2 & B_2 \\
& \ddots & \ddots & \ddots \\
& & A_{p-1} & C_{p-1} & B_{p-1} \\
& & & A_p & C_p
\end{pmatrix}
\begin{pmatrix}
x_1 \\
\vdots \\
x_p \\
\xi_i \\
\vdots \\
\xi_{p-1}
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_{p-1} \\
b_p
\end{pmatrix},
\]

(5.2)

We assume to have \( p \) processors available. The \( i \)-th processor holds \( A_i, B_i, C_i, \) and \( b_i \).

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

$$
\begin{pmatrix}
R_1 & E_1 & E_2 & \cdots & \cdots & E_{p-1} & E_p \\
R_2 & F_2 & & & & & \\
\vdots & & & & & & \\
R_{p-1} & & & & & & \\
R_p & F_p & & & & & \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_p \\
\xi_1 \\
\xi_2 \\
\vdots \\
\xi_{p-1}
\end{pmatrix}
=
\begin{pmatrix}
e_1 \\
e_2 \\
\vdots \\
e_p
\end{pmatrix},
$$

(5.3)

where

$$E_i = L_i^{-1} P_i B_i, \quad F_i = L_i^{-1} P_i C_i, \quad e_i = L_i^{-1} P_i b_i.$$ 

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

$$C_{\text{step}} = C_{\text{LUP}}(\frac{n}{p}, k, k) + (p-1)C_{\text{LUP}}(\frac{n}{p}, 2k, 0) + \left(\frac{2k+1}{p}\right) \left(6k^2 + 6k + 1\right) + k \left(86k^2 - 27k + 13\right) \text{ flops}.$$ 

Notice, that we actually factor a lower triangular matrices if we image $B_i$ 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

$$C_{\text{par}}^\text{step} = C_{\text{LUP}}(\frac{n}{p}, k, k) + (2k+1)C_{\text{torw}}(\frac{n}{p}, 2k)$$

$$= 2k(8k + 3) \frac{n}{p} - \frac{k}{3} (2k + 1)(28k - 5) \text{ flops}.$$ 

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:

$$R_i = \begin{bmatrix}
\hat{R}_i \\
0
\end{bmatrix}, \quad E_i = \begin{bmatrix}
\hat{E}_{i1} \\
\hat{E}_{i2}
\end{bmatrix}, \quad F_i = \begin{bmatrix}
\hat{F}_{i1} \\
\hat{F}_{i2}
\end{bmatrix}, \quad e_i = \begin{bmatrix}
\hat{e}_{i1} \\
\hat{e}_{i2}
\end{bmatrix}.$$ 

(5.4)

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

$$\begin{pmatrix}
\hat{E}_{12} \\
\hat{F}_{22} & \hat{E}_{22} \\
\ddots & \ddots & \ddots \\
\hat{F}_{p-1,2} & \hat{E}_{p-1,2} \\
\hat{F}_{p2}
\end{pmatrix} \begin{bmatrix}
\xi_1 \\
\vdots \\
\xi_{p-1}
\end{bmatrix} = \begin{bmatrix}
\hat{e}_{12} \\
\vdots \\
\hat{e}_{p2}
\end{bmatrix}.$$ 

(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 $\hat{F}_{12}$, 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

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

Because this step is performed sequentially, the parallel complexity is \( C_{\text{step}2} \) plus the complexity of the communication,

\[
C_{\text{step}2}^\text{par} = (p - 1) \left( \frac{2k}{3} (13k^3 + 12k + 5) + 2\sigma + (2k^2 + 2k) \tau \right) - 2k^2(3k + 2) \text{ flops.}
\]

3. Back substitution

Knowing the vectors \( \xi_i, 1 \leq i < p \), the \( i \)-th processor can compute its section of \( \mathbf{x} \) by

\[
\begin{align*}
\mathbf{x}_1 &= \hat{R}_d^{-1}(\hat{e}_{11} - \hat{E}_{11}\xi_1), \\
\mathbf{x}_i &= \hat{R}_d^{-1}(\hat{e}_{ii} - \hat{E}_{ii}\xi_{i-1} - \hat{E}_{i1}\xi_i), & 1 < i < p, \\
\mathbf{x}_p &= \hat{R}_d^{-1}(\hat{e}_{p1} - \hat{E}_{p1}\xi_{p-1}).
\end{align*}
\]

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

\[
C_{\text{step}3} = \left( 8k - 1 - \frac{4k}{p} \right) n - 2k(2k + 1) \quad \text{and} \quad C_{\text{step}3}^\text{par} = (8k - 1) \frac{n}{p},
\]

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

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

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

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

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

The parallel complexity of the double width separator approach is

\[
C_{\text{dws}}^\text{par} \approx 16k^2 \frac{n}{p} + \left( \frac{26}{3} k^3 + 2\sigma + 2k^2 \tau \right) p \text{ flops.}
\]

Thus, speedup becomes

\[
S_{\text{dws}}(n, k, p) = \frac{C_{\text{GaussP}}(n, k)}{C_{\text{dws}}^\text{par}(n, k, p)} \approx 4 + \left( \frac{12k}{p} + \frac{1}{2} \tau + \frac{\sigma}{2n} \right) \frac{1}{p \tau},
\]

yielding optimal processor number

\[
p_{\text{dws}}^\text{opt} \approx \sqrt{\frac{24n}{13k}} \sqrt{\frac{1}{1 + \frac{3\tau}{12n} + \frac{3\tau}{13k}}} \]

and corresponding optimal speedup

\[
S_{\text{dws}}^\text{opt} \approx \sqrt{\frac{3n}{104k}} \sqrt{\frac{1}{1 + \frac{3\tau}{12n} + \frac{3\tau}{13k}}},
\]

19
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 $p_{\text{opt}}$ of processors. $p_{\text{opt}}$ depends on the size of the problem. The existence of $p_{\text{opt}}$ means that these algorithms are not scalable. In Tab. 6.1 we list the speedup, the optimal number of processors and corresponding maximal speedsups for the various algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speedup $S(n, k, \sigma, \tau)$</th>
<th>$p_{\text{opt}}$</th>
<th>$S_{\text{opt}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>d&amp;c</td>
<td>$p$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
<tr>
<td>d&amp;c with cr</td>
<td>$\frac{4}{3k} + \frac{\tau}{2} + \frac{\sigma}{k^2}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
<tr>
<td>sws</td>
<td>$\frac{4}{3k} + \frac{\tau}{2} + \frac{\sigma}{k^2}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
<tr>
<td>sws with cr</td>
<td>$\frac{4}{3k} + \frac{\tau}{2} + \frac{\sigma}{k^2}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
<tr>
<td>symmetric sws</td>
<td>$\frac{4}{3k} + \frac{\tau}{2} + \frac{\sigma}{k^2}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
<tr>
<td>dws</td>
<td>$\frac{4}{3k} + \frac{\tau}{2} + \frac{\sigma}{k^2}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
<tr>
<td>symmetric dws</td>
<td>$\frac{4}{3k} + \frac{\tau}{2} + \frac{\sigma}{k^2}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
<td>$\frac{12n}{7k} + \frac{2\tau}{12k} + \frac{\sigma}{13k}$</td>
</tr>
</tbody>
</table>

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 contains a $p^3$ 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 ssw and ssw/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

\[
E(n, k, p) := \frac{S(n, k, p)}{p}.
\]  

(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

\[
p^2 = n \left( \frac{7}{3} \left[ \frac{k}{\tau} + \frac{\sigma}{k^2} \right] \right)
\]

for sws and

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

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.

<table>
<thead>
<tr>
<th>$p$</th>
<th>sws</th>
<th>sws</th>
<th>sws/cr</th>
<th>sws/cr</th>
<th>$p$</th>
<th>sws</th>
<th>sws</th>
<th>sws/cr</th>
<th>sws/cr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IEEE</td>
<td>no IEEE</td>
<td>IEEE</td>
<td>no IEEE</td>
<td></td>
<td>IEEE</td>
<td>no IEEE</td>
<td>IEEE</td>
<td>no IEEE</td>
</tr>
<tr>
<td>1</td>
<td>612</td>
<td>608</td>
<td>615</td>
<td>609</td>
<td>25</td>
<td>387</td>
<td>384</td>
<td>297</td>
<td>225</td>
</tr>
<tr>
<td>2</td>
<td>3125</td>
<td>648</td>
<td>4282</td>
<td>658</td>
<td>26</td>
<td>391</td>
<td>394</td>
<td>293</td>
<td>225</td>
</tr>
<tr>
<td>3</td>
<td>3055</td>
<td>511</td>
<td>4138</td>
<td>521</td>
<td>27</td>
<td>397</td>
<td>396</td>
<td>291</td>
<td>222</td>
</tr>
<tr>
<td>4</td>
<td>3057</td>
<td>433</td>
<td>4064</td>
<td>436</td>
<td>28</td>
<td>402</td>
<td>401</td>
<td>293</td>
<td>221</td>
</tr>
<tr>
<td>5</td>
<td>3103</td>
<td>391</td>
<td>4010</td>
<td>381</td>
<td>29</td>
<td>407</td>
<td>407</td>
<td>292</td>
<td>220</td>
</tr>
<tr>
<td>6</td>
<td>3158</td>
<td>375</td>
<td>1502</td>
<td>362</td>
<td>30</td>
<td>414</td>
<td>415</td>
<td>287</td>
<td>220</td>
</tr>
<tr>
<td>7</td>
<td>3200</td>
<td>367</td>
<td>1193</td>
<td>336</td>
<td>31</td>
<td>419</td>
<td>420</td>
<td>285</td>
<td>216</td>
</tr>
<tr>
<td>8</td>
<td>1315</td>
<td>357</td>
<td>952</td>
<td>309</td>
<td>32</td>
<td>428</td>
<td>425</td>
<td>286</td>
<td>217</td>
</tr>
<tr>
<td>9</td>
<td>1140</td>
<td>346</td>
<td>757</td>
<td>299</td>
<td>33</td>
<td>432</td>
<td>432</td>
<td>292</td>
<td>220</td>
</tr>
<tr>
<td>10</td>
<td>1006</td>
<td>335</td>
<td>604</td>
<td>285</td>
<td>34</td>
<td>440</td>
<td>438</td>
<td>294</td>
<td>218</td>
</tr>
<tr>
<td>11</td>
<td>894</td>
<td>335</td>
<td>318</td>
<td>273</td>
<td>35</td>
<td>447</td>
<td>444</td>
<td>288</td>
<td>218</td>
</tr>
<tr>
<td>12</td>
<td>821</td>
<td>337</td>
<td>314</td>
<td>270</td>
<td>36</td>
<td>451</td>
<td>453</td>
<td>288</td>
<td>221</td>
</tr>
<tr>
<td>13</td>
<td>750</td>
<td>336</td>
<td>331</td>
<td>262</td>
<td>37</td>
<td>456</td>
<td>458</td>
<td>288</td>
<td>217</td>
</tr>
<tr>
<td>14</td>
<td>699</td>
<td>336</td>
<td>325</td>
<td>255</td>
<td>38</td>
<td>465</td>
<td>465</td>
<td>286</td>
<td>213</td>
</tr>
<tr>
<td>15</td>
<td>799</td>
<td>340</td>
<td>319</td>
<td>251</td>
<td>39</td>
<td>465</td>
<td>471</td>
<td>281</td>
<td>215</td>
</tr>
<tr>
<td>16</td>
<td>380</td>
<td>344</td>
<td>312</td>
<td>245</td>
<td>40</td>
<td>478</td>
<td>473</td>
<td>279</td>
<td>212</td>
</tr>
<tr>
<td>17</td>
<td>345</td>
<td>347</td>
<td>319</td>
<td>245</td>
<td>41</td>
<td>484</td>
<td>483</td>
<td>282</td>
<td>211</td>
</tr>
<tr>
<td>18</td>
<td>348</td>
<td>347</td>
<td>316</td>
<td>245</td>
<td>42</td>
<td>495</td>
<td>488</td>
<td>731</td>
<td>212</td>
</tr>
<tr>
<td>19</td>
<td>353</td>
<td>353</td>
<td>314</td>
<td>239</td>
<td>43</td>
<td>497</td>
<td>497</td>
<td>477</td>
<td>211</td>
</tr>
<tr>
<td>20</td>
<td>351</td>
<td>355</td>
<td>306</td>
<td>236</td>
<td>44</td>
<td>508</td>
<td>504</td>
<td>259</td>
<td>210</td>
</tr>
<tr>
<td>21</td>
<td>364</td>
<td>364</td>
<td>758</td>
<td>233</td>
<td>45</td>
<td>512</td>
<td>508</td>
<td>258</td>
<td>212</td>
</tr>
<tr>
<td>22</td>
<td>364</td>
<td>365</td>
<td>276</td>
<td>227</td>
<td>46</td>
<td>517</td>
<td>520</td>
<td>256</td>
<td>212</td>
</tr>
<tr>
<td>23</td>
<td>373</td>
<td>368</td>
<td>272</td>
<td>225</td>
<td>47</td>
<td>523</td>
<td>527</td>
<td>251</td>
<td>207</td>
</tr>
<tr>
<td>24</td>
<td>382</td>
<td>379</td>
<td>271</td>
<td>227</td>
<td>48</td>
<td>538</td>
<td>536</td>
<td>263</td>
<td>210</td>
</tr>
</tbody>
</table>

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

22
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 \((n, k) = (80000, 40)\). 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.
<table>
<thead>
<tr>
<th>p</th>
<th>sws</th>
<th>sws</th>
<th>sws/ct</th>
<th>sws/ct</th>
<th>p</th>
<th>sws</th>
<th>sws</th>
<th>sws/ct</th>
<th>sws/ct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IEEE</td>
<td>no IEEE</td>
<td></td>
<td></td>
<td></td>
<td>IEEE</td>
<td>no IEEE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5806</td>
<td>5796</td>
<td>5858</td>
<td>5797</td>
<td>49</td>
<td>9704</td>
<td>851</td>
<td>4143</td>
<td>443</td>
</tr>
<tr>
<td>2</td>
<td>7966</td>
<td>5498</td>
<td>9200</td>
<td>5526</td>
<td>50</td>
<td>9845</td>
<td>846</td>
<td>4143</td>
<td>442</td>
</tr>
<tr>
<td>3</td>
<td>6314</td>
<td>3777</td>
<td>7442</td>
<td>3768</td>
<td>51</td>
<td>13194</td>
<td>864</td>
<td>4225</td>
<td>441</td>
</tr>
<tr>
<td>4</td>
<td>5533</td>
<td>2908</td>
<td>6550</td>
<td>2877</td>
<td>52</td>
<td>13294</td>
<td>833</td>
<td>3516</td>
<td>433</td>
</tr>
<tr>
<td>5</td>
<td>5086</td>
<td>2529</td>
<td>6052</td>
<td>2336</td>
<td>53</td>
<td>9623</td>
<td>842</td>
<td>2618</td>
<td>435</td>
</tr>
<tr>
<td>6</td>
<td>4849</td>
<td>2061</td>
<td>5669</td>
<td>1999</td>
<td>54</td>
<td>3116</td>
<td>870</td>
<td>1898</td>
<td>425</td>
</tr>
<tr>
<td>7</td>
<td>4671</td>
<td>1823</td>
<td>5421</td>
<td>1743</td>
<td>55</td>
<td>2843</td>
<td>879</td>
<td>1816</td>
<td>423</td>
</tr>
<tr>
<td>8</td>
<td>4561</td>
<td>1641</td>
<td>5236</td>
<td>1545</td>
<td>56</td>
<td>2830</td>
<td>867</td>
<td>1763</td>
<td>424</td>
</tr>
<tr>
<td>9</td>
<td>4488</td>
<td>1501</td>
<td>5098</td>
<td>1406</td>
<td>57</td>
<td>2776</td>
<td>880</td>
<td>1721</td>
<td>413</td>
</tr>
<tr>
<td>10</td>
<td>4468</td>
<td>1394</td>
<td>4980</td>
<td>1282</td>
<td>58</td>
<td>2784</td>
<td>873</td>
<td>1681</td>
<td>410</td>
</tr>
<tr>
<td>11</td>
<td>4352</td>
<td>1301</td>
<td>4886</td>
<td>1183</td>
<td>59</td>
<td>2742</td>
<td>858</td>
<td>1642</td>
<td>410</td>
</tr>
<tr>
<td>12</td>
<td>4434</td>
<td>1238</td>
<td>4792</td>
<td>1109</td>
<td>60</td>
<td>2726</td>
<td>856</td>
<td>1600</td>
<td>403</td>
</tr>
<tr>
<td>13</td>
<td>4473</td>
<td>1177</td>
<td>4740</td>
<td>1036</td>
<td>61</td>
<td>2686</td>
<td>852</td>
<td>1557</td>
<td>399</td>
</tr>
<tr>
<td>14</td>
<td>4483</td>
<td>1130</td>
<td>4672</td>
<td>983</td>
<td>62</td>
<td>2700</td>
<td>856</td>
<td>1530</td>
<td>403</td>
</tr>
<tr>
<td>15</td>
<td>4721</td>
<td>1080</td>
<td>4625</td>
<td>933</td>
<td>63</td>
<td>2672</td>
<td>858</td>
<td>1490</td>
<td>392</td>
</tr>
<tr>
<td>16</td>
<td>4651</td>
<td>1046</td>
<td>4572</td>
<td>884</td>
<td>64</td>
<td>2649</td>
<td>871</td>
<td>1453</td>
<td>388</td>
</tr>
<tr>
<td>17</td>
<td>4693</td>
<td>1015</td>
<td>4541</td>
<td>854</td>
<td>65</td>
<td>2655</td>
<td>868</td>
<td>1432</td>
<td>398</td>
</tr>
<tr>
<td>18</td>
<td>4642</td>
<td>1059</td>
<td>4508</td>
<td>821</td>
<td>66</td>
<td>2634</td>
<td>866</td>
<td>1414</td>
<td>392</td>
</tr>
<tr>
<td>19</td>
<td>4685</td>
<td>964</td>
<td>4476</td>
<td>785</td>
<td>67</td>
<td>2655</td>
<td>876</td>
<td>1356</td>
<td>386</td>
</tr>
<tr>
<td>20</td>
<td>4750</td>
<td>944</td>
<td>4447</td>
<td>753</td>
<td>68</td>
<td>2644</td>
<td>882</td>
<td>1352</td>
<td>398</td>
</tr>
<tr>
<td>21</td>
<td>4808</td>
<td>1007</td>
<td>4421</td>
<td>728</td>
<td>69</td>
<td>2638</td>
<td>882</td>
<td>1320</td>
<td>399</td>
</tr>
<tr>
<td>22</td>
<td>4846</td>
<td>908</td>
<td>4396</td>
<td>706</td>
<td>70</td>
<td>2633</td>
<td>893</td>
<td>1290</td>
<td>382</td>
</tr>
<tr>
<td>23</td>
<td>4919</td>
<td>901</td>
<td>4373</td>
<td>685</td>
<td>71</td>
<td>2624</td>
<td>895</td>
<td>1263</td>
<td>376</td>
</tr>
<tr>
<td>24</td>
<td>4966</td>
<td>906</td>
<td>4367</td>
<td>687</td>
<td>72</td>
<td>2618</td>
<td>897</td>
<td>1234</td>
<td>390</td>
</tr>
<tr>
<td>25</td>
<td>5043</td>
<td>871</td>
<td>4357</td>
<td>652</td>
<td>73</td>
<td>2610</td>
<td>909</td>
<td>1193</td>
<td>375</td>
</tr>
<tr>
<td>26</td>
<td>5091</td>
<td>870</td>
<td>4323</td>
<td>641</td>
<td>74</td>
<td>2600</td>
<td>911</td>
<td>1162</td>
<td>370</td>
</tr>
<tr>
<td>27</td>
<td>5155</td>
<td>860</td>
<td>4311</td>
<td>622</td>
<td>75</td>
<td>2591</td>
<td>913</td>
<td>1157</td>
<td>379</td>
</tr>
<tr>
<td>28</td>
<td>5807</td>
<td>848</td>
<td>4300</td>
<td>606</td>
<td>76</td>
<td>2590</td>
<td>918</td>
<td>1109</td>
<td>365</td>
</tr>
<tr>
<td>29</td>
<td>5522</td>
<td>844</td>
<td>4295</td>
<td>597</td>
<td>77</td>
<td>2584</td>
<td>920</td>
<td>1089</td>
<td>363</td>
</tr>
<tr>
<td>30</td>
<td>5363</td>
<td>839</td>
<td>4273</td>
<td>584</td>
<td>78</td>
<td>2578</td>
<td>931</td>
<td>1065</td>
<td>369</td>
</tr>
<tr>
<td>31</td>
<td>5523</td>
<td>828</td>
<td>4273</td>
<td>563</td>
<td>79</td>
<td>2564</td>
<td>952</td>
<td>1061</td>
<td>371</td>
</tr>
<tr>
<td>32</td>
<td>5497</td>
<td>834</td>
<td>4250</td>
<td>559</td>
<td>80</td>
<td>2577</td>
<td>961</td>
<td>1017</td>
<td>375</td>
</tr>
<tr>
<td>33</td>
<td>5906</td>
<td>832</td>
<td>4255</td>
<td>572</td>
<td>81</td>
<td>2562</td>
<td>963</td>
<td>1017</td>
<td>365</td>
</tr>
<tr>
<td>34</td>
<td>5741</td>
<td>829</td>
<td>4247</td>
<td>562</td>
<td>82</td>
<td>2563</td>
<td>961</td>
<td>995</td>
<td>356</td>
</tr>
<tr>
<td>35</td>
<td>5799</td>
<td>828</td>
<td>4235</td>
<td>538</td>
<td>83</td>
<td>2546</td>
<td>973</td>
<td>956</td>
<td>350</td>
</tr>
<tr>
<td>36</td>
<td>5883</td>
<td>826</td>
<td>4218</td>
<td>528</td>
<td>84</td>
<td>2565</td>
<td>975</td>
<td>955</td>
<td>347</td>
</tr>
<tr>
<td>37</td>
<td>5947</td>
<td>824</td>
<td>4221</td>
<td>532</td>
<td>85</td>
<td>2556</td>
<td>990</td>
<td>914</td>
<td>345</td>
</tr>
<tr>
<td>38</td>
<td>6604</td>
<td>829</td>
<td>4209</td>
<td>519</td>
<td>86</td>
<td>2560</td>
<td>991</td>
<td>915</td>
<td>356</td>
</tr>
<tr>
<td>39</td>
<td>6088</td>
<td>824</td>
<td>4189</td>
<td>506</td>
<td>87</td>
<td>2548</td>
<td>991</td>
<td>898</td>
<td>345</td>
</tr>
<tr>
<td>40</td>
<td>6231</td>
<td>818</td>
<td>4182</td>
<td>519</td>
<td>88</td>
<td>2558</td>
<td>1001</td>
<td>855</td>
<td>359</td>
</tr>
<tr>
<td>41</td>
<td>6315</td>
<td>847</td>
<td>4194</td>
<td>492</td>
<td>89</td>
<td>2559</td>
<td>1008</td>
<td>841</td>
<td>338</td>
</tr>
<tr>
<td>42</td>
<td>6461</td>
<td>832</td>
<td>4178</td>
<td>487</td>
<td>90</td>
<td>2563</td>
<td>1020</td>
<td>843</td>
<td>342</td>
</tr>
<tr>
<td>43</td>
<td>6609</td>
<td>837</td>
<td>4175</td>
<td>480</td>
<td>91</td>
<td>2564</td>
<td>1024</td>
<td>800</td>
<td>334</td>
</tr>
<tr>
<td>44</td>
<td>6553</td>
<td>837</td>
<td>4160</td>
<td>476</td>
<td>92</td>
<td>2582</td>
<td>1019</td>
<td>804</td>
<td>334</td>
</tr>
<tr>
<td>45</td>
<td>6538</td>
<td>837</td>
<td>4164</td>
<td>467</td>
<td>93</td>
<td>2589</td>
<td>1031</td>
<td>789</td>
<td>332</td>
</tr>
<tr>
<td>46</td>
<td>6694</td>
<td>841</td>
<td>4154</td>
<td>463</td>
<td>94</td>
<td>2582</td>
<td>1041</td>
<td>752</td>
<td>331</td>
</tr>
<tr>
<td>47</td>
<td>6994</td>
<td>843</td>
<td>4157</td>
<td>455</td>
<td>95</td>
<td>2581</td>
<td>1041</td>
<td>738</td>
<td>341</td>
</tr>
<tr>
<td>48</td>
<td>9651</td>
<td>857</td>
<td>4146</td>
<td>447</td>
<td>96</td>
<td>2585</td>
<td>1050</td>
<td>741</td>
<td>326</td>
</tr>
</tbody>
</table>

Table 6.3: Times in milliseconds for sws and sws/ct for the problem size \((n, k) = (100000, 10)\).
<table>
<thead>
<tr>
<th>p</th>
<th>sws IEEE</th>
<th>sws no IEEE</th>
<th>sws/cr IEEE</th>
<th>sws/cr no IEEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46368*</td>
<td>46368*</td>
<td>46368*</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>16078</td>
<td>11611</td>
<td>14926</td>
<td>10991</td>
</tr>
<tr>
<td>9</td>
<td>14971</td>
<td>10443</td>
<td>13645</td>
<td>9798</td>
</tr>
<tr>
<td>10</td>
<td>14153</td>
<td>9521</td>
<td>12597</td>
<td>8839</td>
</tr>
<tr>
<td>11</td>
<td>13497</td>
<td>8755</td>
<td>11798</td>
<td>8054</td>
</tr>
<tr>
<td>12</td>
<td>13024</td>
<td>8121</td>
<td>11129</td>
<td>7409</td>
</tr>
<tr>
<td>13</td>
<td>12502</td>
<td>7587</td>
<td>10693</td>
<td>6856</td>
</tr>
<tr>
<td>14</td>
<td>12172</td>
<td>7129</td>
<td>10100</td>
<td>6387</td>
</tr>
<tr>
<td>15</td>
<td>11848</td>
<td>6740</td>
<td>9712</td>
<td>5978</td>
</tr>
<tr>
<td>16</td>
<td>11639</td>
<td>6402</td>
<td>9323</td>
<td>5618</td>
</tr>
<tr>
<td>17</td>
<td>11422</td>
<td>6099</td>
<td>9019</td>
<td>5360</td>
</tr>
<tr>
<td>18</td>
<td>11273</td>
<td>5832</td>
<td>8754</td>
<td>5028</td>
</tr>
<tr>
<td>19</td>
<td>11233</td>
<td>5582</td>
<td>8468</td>
<td>4777</td>
</tr>
<tr>
<td>20</td>
<td>10987</td>
<td>5364</td>
<td>8269</td>
<td>4549</td>
</tr>
<tr>
<td>21</td>
<td>10910</td>
<td>5165</td>
<td>8033</td>
<td>4342</td>
</tr>
<tr>
<td>22</td>
<td>10875</td>
<td>4984</td>
<td>7876</td>
<td>4154</td>
</tr>
<tr>
<td>23</td>
<td>10784</td>
<td>4838</td>
<td>7705</td>
<td>3986</td>
</tr>
<tr>
<td>24</td>
<td>10766</td>
<td>4704</td>
<td>7533</td>
<td>3832</td>
</tr>
<tr>
<td>25</td>
<td>10714</td>
<td>4566</td>
<td>7402</td>
<td>3686</td>
</tr>
<tr>
<td>26</td>
<td>10719</td>
<td>4451</td>
<td>7305</td>
<td>3554</td>
</tr>
<tr>
<td>27</td>
<td>10686</td>
<td>4316</td>
<td>7146</td>
<td>3430</td>
</tr>
<tr>
<td>28</td>
<td>10733</td>
<td>4212</td>
<td>7025</td>
<td>3317</td>
</tr>
<tr>
<td>29</td>
<td>10869</td>
<td>4112</td>
<td>7033</td>
<td>3212</td>
</tr>
<tr>
<td>30</td>
<td>10753</td>
<td>4041</td>
<td>6814</td>
<td>3113</td>
</tr>
<tr>
<td>31</td>
<td>10777</td>
<td>3944</td>
<td>6728</td>
<td>3026</td>
</tr>
<tr>
<td>32</td>
<td>10802</td>
<td>3886</td>
<td>6635</td>
<td>2937</td>
</tr>
<tr>
<td>33</td>
<td>10804</td>
<td>3779</td>
<td>6560</td>
<td>2860</td>
</tr>
<tr>
<td>34</td>
<td>10874</td>
<td>3719</td>
<td>6590</td>
<td>2790</td>
</tr>
<tr>
<td>35</td>
<td>10875</td>
<td>3666</td>
<td>6412</td>
<td>2790</td>
</tr>
<tr>
<td>36</td>
<td>10935</td>
<td>3590</td>
<td>6343</td>
<td>2648</td>
</tr>
<tr>
<td>37</td>
<td>10964</td>
<td>3535</td>
<td>6273</td>
<td>2638</td>
</tr>
<tr>
<td>38</td>
<td>10935</td>
<td>3480</td>
<td>6208</td>
<td>2517</td>
</tr>
<tr>
<td>39</td>
<td>10957</td>
<td>3431</td>
<td>6157</td>
<td>2455</td>
</tr>
<tr>
<td>40</td>
<td>11128</td>
<td>3391</td>
<td>6102</td>
<td>2403</td>
</tr>
<tr>
<td>41</td>
<td>11186</td>
<td>3350</td>
<td>6036</td>
<td>2354</td>
</tr>
<tr>
<td>42</td>
<td>11253</td>
<td>3326</td>
<td>5995</td>
<td>2301</td>
</tr>
<tr>
<td>43</td>
<td>11300</td>
<td>3265</td>
<td>5936</td>
<td>2252</td>
</tr>
<tr>
<td>44</td>
<td>11397</td>
<td>3241</td>
<td>5904</td>
<td>2210</td>
</tr>
<tr>
<td>45</td>
<td>11433</td>
<td>3190</td>
<td>5855</td>
<td>2166</td>
</tr>
<tr>
<td>46</td>
<td>11553</td>
<td>3171</td>
<td>5812</td>
<td>2122</td>
</tr>
<tr>
<td>47</td>
<td>11583</td>
<td>3117</td>
<td>5776</td>
<td>2078</td>
</tr>
<tr>
<td>48</td>
<td>11696</td>
<td>3119</td>
<td>5745</td>
<td>2043</td>
</tr>
<tr>
<td>49</td>
<td>11810</td>
<td>3073</td>
<td>5704</td>
<td>2005</td>
</tr>
<tr>
<td>50</td>
<td>11889</td>
<td>3030</td>
<td>5674</td>
<td>1972</td>
</tr>
<tr>
<td>51</td>
<td>11949</td>
<td>3016</td>
<td>5634</td>
<td>1937</td>
</tr>
</tbody>
</table>

Table 6.4: Times in milliseconds for sws and sws/cr for the problem size \((n, k) = (800000, 10)\).
<table>
<thead>
<tr>
<th>( p )</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
<th>sws</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>180000*</td>
<td>180000*</td>
<td>180000*</td>
<td>60</td>
<td>276228</td>
<td>12124</td>
<td>178552</td>
<td>9884</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>226844</td>
<td>22411</td>
<td>184503</td>
<td>20439</td>
<td>61</td>
<td>275458</td>
<td>12041</td>
<td>176584</td>
<td>8831</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>231692</td>
<td>21683</td>
<td>187147</td>
<td>19662</td>
<td>62</td>
<td>280148</td>
<td>11951</td>
<td>194330</td>
<td>8721</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>233003</td>
<td>21160</td>
<td>184480</td>
<td>18993</td>
<td>63</td>
<td>277826</td>
<td>11866</td>
<td>177127</td>
<td>8584</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>230277</td>
<td>20463</td>
<td>186144</td>
<td>18363</td>
<td>64</td>
<td>278210</td>
<td>11781</td>
<td>177955</td>
<td>8480</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>231328</td>
<td>19885</td>
<td>184049</td>
<td>17691</td>
<td>65</td>
<td>283594</td>
<td>11697</td>
<td>177978</td>
<td>8414</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>230978</td>
<td>19288</td>
<td>183907</td>
<td>17140</td>
<td>66</td>
<td>282490</td>
<td>11627</td>
<td>172724</td>
<td>8315</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>234083</td>
<td>18967</td>
<td>182190</td>
<td>16581</td>
<td>67</td>
<td>283695</td>
<td>11546</td>
<td>176810</td>
<td>8206</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>236343</td>
<td>18346</td>
<td>182589</td>
<td>16086</td>
<td>68</td>
<td>284913</td>
<td>11478</td>
<td>178497</td>
<td>8103</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>234138</td>
<td>17900</td>
<td>181003</td>
<td>15631</td>
<td>69</td>
<td>290630</td>
<td>11402</td>
<td>175562</td>
<td>8012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>237303</td>
<td>17499</td>
<td>182652</td>
<td>15248</td>
<td>70</td>
<td>288933</td>
<td>11359</td>
<td>176733</td>
<td>7965</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>240290</td>
<td>17121</td>
<td>182476</td>
<td>14842</td>
<td>71</td>
<td>293822</td>
<td>11290</td>
<td>175385</td>
<td>7820</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>242065</td>
<td>16785</td>
<td>183876</td>
<td>14442</td>
<td>72</td>
<td>292764</td>
<td>11247</td>
<td>180244</td>
<td>7726</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>242193</td>
<td>16446</td>
<td>181550</td>
<td>14085</td>
<td>73</td>
<td>297453</td>
<td>11171</td>
<td>176878</td>
<td>7649</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>242828</td>
<td>16137</td>
<td>180058</td>
<td>13746</td>
<td>74</td>
<td>300384</td>
<td>11146</td>
<td>176583</td>
<td>7571</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>244528</td>
<td>15853</td>
<td>180332</td>
<td>13409</td>
<td>75</td>
<td>298085</td>
<td>11085</td>
<td>176912</td>
<td>7484</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>245552</td>
<td>15561</td>
<td>180281</td>
<td>13092</td>
<td>76</td>
<td>299822</td>
<td>11046</td>
<td>176364</td>
<td>7393</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>241679</td>
<td>15289</td>
<td>179491</td>
<td>12814</td>
<td>77</td>
<td>297281</td>
<td>11013</td>
<td>178067</td>
<td>7314</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>246965</td>
<td>15047</td>
<td>183271</td>
<td>12518</td>
<td>78</td>
<td>330815</td>
<td>10954</td>
<td>175096</td>
<td>7232</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>249553</td>
<td>14826</td>
<td>183510</td>
<td>12246</td>
<td>79</td>
<td>460348</td>
<td>10896</td>
<td>182069</td>
<td>7150</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>249418</td>
<td>14611</td>
<td>179230</td>
<td>11990</td>
<td>80</td>
<td>495903</td>
<td>10876</td>
<td>179871</td>
<td>7092</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>250911</td>
<td>14394</td>
<td>179165</td>
<td>11743</td>
<td>81</td>
<td>532276</td>
<td>10706</td>
<td>175895</td>
<td>7018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>255827</td>
<td>14185</td>
<td>178057</td>
<td>11519</td>
<td>82</td>
<td>388967</td>
<td>10672</td>
<td>176166</td>
<td>6952</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>255093</td>
<td>13997</td>
<td>178723</td>
<td>11287</td>
<td>83</td>
<td>360599</td>
<td>10740</td>
<td>174826</td>
<td>6875</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>254598</td>
<td>13823</td>
<td>178592</td>
<td>11063</td>
<td>84</td>
<td>328344</td>
<td>10725</td>
<td>170723</td>
<td>6738</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>255689</td>
<td>13658</td>
<td>178415</td>
<td>10878</td>
<td>85</td>
<td>330594</td>
<td>10677</td>
<td>170576</td>
<td>6750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>262859</td>
<td>13490</td>
<td>178375</td>
<td>10691</td>
<td>86</td>
<td>330469</td>
<td>10661</td>
<td>170541</td>
<td>6693</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>259417</td>
<td>13316</td>
<td>178172</td>
<td>10504</td>
<td>87</td>
<td>330785</td>
<td>10619</td>
<td>170399</td>
<td>6696</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>261530</td>
<td>13182</td>
<td>177579</td>
<td>10319</td>
<td>88</td>
<td>330404</td>
<td>10605</td>
<td>170413</td>
<td>6567</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>262061</td>
<td>13053</td>
<td>197348</td>
<td>10136</td>
<td>89</td>
<td>336000</td>
<td>10580</td>
<td>170273</td>
<td>6565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>262293</td>
<td>12912</td>
<td>172757</td>
<td>9985</td>
<td>90</td>
<td>340024</td>
<td>10404</td>
<td>170334</td>
<td>6348</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>267280</td>
<td>12776</td>
<td>172477</td>
<td>9804</td>
<td>91</td>
<td>338540</td>
<td>10520</td>
<td>170168</td>
<td>6381</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>267727</td>
<td>12679</td>
<td>177687</td>
<td>9653</td>
<td>92</td>
<td>345062</td>
<td>10414</td>
<td>170226</td>
<td>6331</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>265153</td>
<td>12556</td>
<td>177645</td>
<td>9505</td>
<td>93</td>
<td>345814</td>
<td>10501</td>
<td>170017</td>
<td>6274</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>269418</td>
<td>12425</td>
<td>178342</td>
<td>9349</td>
<td>94</td>
<td>343704</td>
<td>10474</td>
<td>170139</td>
<td>6224</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>270875</td>
<td>12311</td>
<td>176771</td>
<td>9211</td>
<td>95</td>
<td>350723</td>
<td>10467</td>
<td>169940</td>
<td>6168</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>271509</td>
<td>12210</td>
<td>176606</td>
<td>9080</td>
<td>96</td>
<td>348364</td>
<td>10458</td>
<td>169333</td>
<td>6121</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5: Times in milliseconds for sws and sws\text{/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 \(p^{\text{opt}}\) 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 \(p^{\text{opt}}\). 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 \cite{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


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


28


