Direct Parallel Algorithms for Banded Linear Systems

Peter Arbenz* Walter Gander* Kevin Gates*

Abstract. We investigate direct algorithms to solve linear banded systems of equations on MIMD multiprocessor computers with distributed memory. We show that it is hard to beat ordinary one-processor Gaussian elimination. Numerical computation results from the Intel Paragon are given.

1. Introduction

In a project on divide and conquer algorithms in numerical linear algebra, the authors studied parallel algorithms to solve systems of linear equations and eigenvalue problems. The latter consisted in a study of the divide and conquer algorithm proposed by Cuppen [4] and stabilized by Sorensen and Tang [11]. This algorithm is evolving as the standard algorithm for solving the symmetric tridiagonal eigenvalue problem on sequential as on parallel computers. In [7], Gates and Arbenz report on the first successful parallel implementation of the algorithm. They observed almost optimal speedups on the Intel Paragon. The accuracy observed is as good as with any other known (fast) algorithm.

The part of the project concerned with systems of linear equations dealt with the direct solution of banded linear systems. In [2], Arbenz and Gander survey the most important algorithms. In this paper we discuss only one of these methods as all share the same algorithmic structure. The selected algorithm, the single width separator algorithm, is suited for solving diagonally dominant or symmetric definite systems of equations.

We restricted our investigations on algorithms for computers with a distributed memory architecture with powerful processing nodes that support the MIMD programming model e.g. the Intel Paragon and workstation clusters. The architecture of such machines makes programming with a coarse grain parallelism necessary for optimal exploitation of the compute power.

2. The Problem

We consider the system of linear equations

$$A\mathbf{x} = \mathbf{b} \tag{2.1}$$

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

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

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.

On serial computers, Gaussian elimination is the method of choice for solving (2.1). The cost for LU factorization, forward and backward substitution is

$$C_{\text{Gauss}}(n,k) = (2k^2+5k-1)\,n - \frac{4}{3}k^3 + \mathcal{O}(k^2) \text{ flops}, \tag{2.2}$$

where $k := \max(r,s)$ and flop denotes a floating point operation $(+,-,\times,/)$.

3. The single-width separator approach

This algorithm has been studied e.g. by Johnsson [10], Dongarra and Johnsson [5], and in modified forms by Wright [12] and Conroy [3]. 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

$$A = \begin{pmatrix} A_1 & B_1 & & & & \\ C_1 & D_1 & C_2 & & & \\ & B_2 & A_2 & B_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & C_{2p-3} & D_{p-1} & C_{2p-2} \\ & & & & B_{2p-2} & A_p \end{pmatrix}, \tag{3.1}$$

*Institute for Scientific Computing, Swiss Federal Institute of Technology (ETH), 8092 Zurich, Switzerland ([arbenz,gander,gates]@inf.ethz.ch)

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \boldsymbol{\xi}_1 \\ \mathbf{x}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \\ \mathbf{x}_p \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \boldsymbol{\beta}_1 \\ \mathbf{b}_2 \\ \vdots \\ \boldsymbol{\beta}_{p-1} \\ \mathbf{b}_p \end{pmatrix},$$

where $A_i \in \mathbb{R}^{n_i \times n_i}$, $B_i, C_i^T \in \mathbb{R}^{n_i \times k}$, $D_i \in \mathbb{R}^{k \times k}$, $\mathbf{x}_i$, $\mathbf{b}_i \in \mathbb{R}^{n_i}$, $\boldsymbol{\xi}_i$, $\boldsymbol{\beta}_i \in \mathbb{R}^k$, $k = \max(r,s)$, 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. 3.1 for the case $p = 4$. The diagonal blocks A_i are band matrices with the same half-band-widths as A.

and

$$\tilde{\mathbf{x}} = \left[\begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \\ \hline \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{array}\right], \quad \tilde{\mathbf{b}} = \left[\begin{array}{c} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{p-1} \\ \mathbf{b}_p \\ \hline \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{p-1} \end{array}\right].$$

The structure of $\tilde{A}$ is depicted in Fig. 3.2.

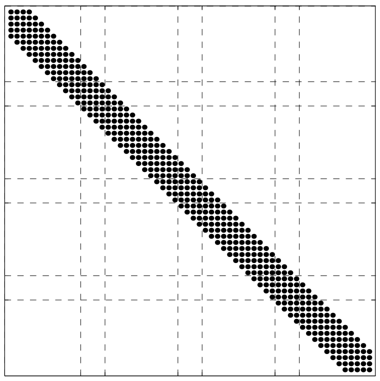


Figure 3.1: Non-zero structure of the original band matrix with $n = 60$, $p = 4$, $n_i = 12$, $r = 4$, and $s = 3$.

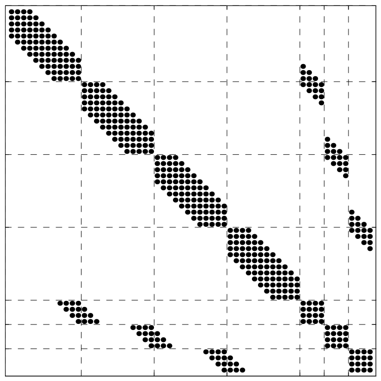


Figure 3.2: Non-zero structure of the block odd-even permuted band matrix with $n = 60$, $p = 4$, $n_i = 12$, $r = 4$, and $s = 3$.

Assume that we have p processors. 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.)

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, (3.1) becomes

$$\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{3.2}$$

where

$$\tilde{A} = \left[\begin{array}{cccc|cccc} A_1 & & & & B_1 & & & \\ & A_2 & & & B_2 & \ddots & & \\ & & \ddots & & & \ddots & B_{2p-3} & \\ & & & A_p & & & B_{2p-2} & \\ \hline C_1 & C_2 & & & D_1 & & & \\ & \ddots & \ddots & & & \ddots & & \\ & & C_{2p-3} & C_{2p-2} & & & D_{p-1} & \end{array}\right],$$

Remark. As matrices appearing in domain decomposition methods have a structure similar to $\tilde{A}$ 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 (3.2), $\tilde{A} = LR$. The structures of the L and R factors are depicted in Fig. 3.3. These are the same structures George [8] obtained with the non-symmetric Cholesky factorization in his one-way dissection scheme. After multiplying (3.2) by L^{-1}, we obtain the system

$$R\tilde{\mathbf{x}} = \tilde{\mathbf{c}} \tag{3.3}$$

where

$$R = \left(\begin{array}{ccccc|cccc} R_1 & & & & & E_1 & & & \\ & R_2 & & & & E_2 & E_3 & & \\ & & \ddots & & & & \ddots & \ddots & \\ & & & R_{p-1} & & & & \ddots & E_{2p-3} \\ & & & & R_p & & & & E_{2p-2} \\ \hline & & & & & T_1 & U_1 & & \\ & & & & & V_2 & T_2 & \ddots & \\ & & & & & & \ddots & \ddots & U_{p-2} \\ & & & & & & & V_{p-1} & T_{p-1} \end{array}\right)$$

$$\tilde{\mathbf{c}}^T = (\mathbf{c}_1^T, \mathbf{c}_2^T, \cdots, \mathbf{c}_{p-1}^T, \mathbf{c}_p^T, \boldsymbol{\gamma}_1^T, \boldsymbol{\gamma}_2, \cdots, \boldsymbol{\gamma}_{p-1}^T)$$

with

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

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 so-called reduced system (3.4),

$$\begin{bmatrix} -F_{2i-2}E_{2i-2} & -F_{2i-2}E_{2i-1} \\ -F_{2i-1}E_{2i-2} & D_i - F_{2i-1}E_{2i-1} \end{bmatrix} \in \mathbb{R}^{2k\times 2k}$$

and

$$\begin{bmatrix} -F_{2i-2}\mathbf{c}_i \\ \boldsymbol{\beta}_i - F_{2i-1}\mathbf{c}_i \end{bmatrix} \in \mathbb{R}^{2k},$$

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

Considering only the highest order terms, the *serial* complexity of this step is

$$C_{\text{step\_1}} \approx \left(8k^2 - \frac{k}{p}6k\right)n - 8pk^3 \text{ flops}.$$

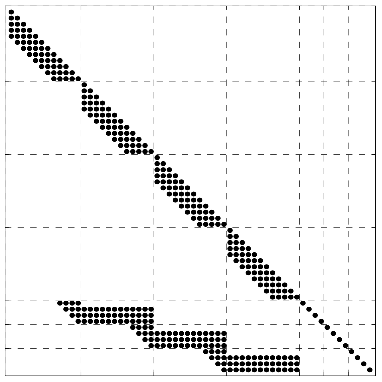
Step 1 is perfectly parallelizable. There is a slight load imbalance in that the first and last processors have less work to do. The *parallel* complexity is

$$C_{\text{step\_1}}^{\text{par}} \approx 8k^2\frac{n}{p} - 8k^3 \text{ flops}.$$

2. Formation and solution of reduced system

The reduced system is

$$\begin{bmatrix} T_1 & U_1 & & & \\ V_2 & T_2 & U_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & U_{p-2} \\ & & & V_{p-1} & T_{p-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-2} \\ \boldsymbol{\xi}_{p-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_{p-2} \\ \boldsymbol{\gamma}_{p-1} \end{bmatrix}. \tag{3.4}$$


(a)

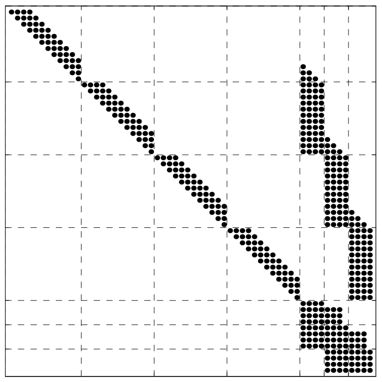

(b)

Figure 3.3: 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 $s = 3$.

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. 3.3b. The reduced system is diagonally dominant and can be solved by block Gaussian elimination or by block cyclic reduction. In both cases, at the beginning of the solution process, the matrices V_i, T_i, U_i and the vector $\boldsymbol{\gamma}_i$ are stored on processor $i+1$.

Block Gaussian elimination of (3.4) costs

$$C_{\text{step\_2}} \approx \frac{14}{3}pk^3 \text{ flops}.$$

For the complexity of the communication, we assume 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 and τ denote the number of flops that can be executed during the message passing startup time and during the transmission of one (8-Byte) floating point number, respectively. The parallel complexity of this step includes the communication complexity of $2(p-1)\sigma + (p-1)(k^2+k)\tau$ flops and thus it becomes

$$C^{\text{par}}_{\text{step\_2}} \approx \frac{14}{3}pk^3 + 2p\sigma + pk^2\tau \text{ flops.}$$

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

$$C_{\text{step\_2,cr}} \approx \frac{38}{3}pk^3 \text{flops}$$

and

$$C^{\text{par}}_{\text{step\_2,cr}} \approx \log_2\lfloor p-1\rfloor\left(\frac{38}{3}pk^3 + 4\sigma + 4k^2\tau\right)\text{flops},$$

respectively.

3. Back substitution

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

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

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

$$C_{\text{step\_3}} \approx 6kn - 7k^2p \text{ flops},$$

and

$$C^{\text{par}}_{\text{step\_3}} \approx 6k\frac{n}{p} - 7k^2 \text{ flops},$$

respectively.

Redundancy and speedup

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

$$C_{\text{sws}}(n,k,p) \approx \left(8k^2 - 6k^2\frac{1}{p}\right)n - \frac{10}{3}pk^3 \text{ flops.} \tag{3.5}$$

Comparing (3.5) with (2.2) we obtain the redundancy [9, p. 113]

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

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

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

speedup becomes

$$\begin{aligned}S_{\text{sws}}(n,k,p) &= \frac{C_{\text{Gauss}}(n,k)}{C^{\text{par}}_{\text{sws}}(n,k,p)}\\ &\approx \frac{p}{4 + \left(\frac{7}{3}k + \frac{1}{2}\tau + \frac{\sigma}{k^2}\right)\frac{p^2}{n}}\end{aligned} \tag{3.6}$$

Clearly, this speedup is far from the ideal speedup $S(p) = p$. 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 sequentially. The latter is the origin of the p^2 term in (3.6) which makes speedup *decrease* as soon as p exceeds a critical number $p^{\text{opt}}_{\text{sws}}$ for which speedup is highest. $p^{\text{opt}}_{\text{sws}}$ is the positive zero of the derivative of S in (3.6),

$$p^{\text{opt}}_{\text{sws}} \approx \sqrt{\frac{12n}{7k}}\sqrt{\frac{1}{1 + \frac{3\tau}{14k} + \frac{3\sigma}{7k^3}}},$$

and yields an optimal speedup of

$$S^{\text{opt}}_{\text{sws}} := S(n,k,p_{\text{opt}}) = \frac{1}{8}p^{\text{opt}}_{\text{sws}}.$$

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

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

whereas the parallel complexity is

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

From this we get the redundancy

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

The speedup is given by

$$\begin{aligned}S_{\text{sws,cr}}(n,k,p) &= \frac{C_{\text{Gauss}}(n,k)}{C^{\text{par}}_{\text{sws,cr}}(n,k,p)}\\ &\approx \frac{p}{4 + \left(\frac{7}{3}k + 2\tau + 2\frac{\sigma}{k^2}\right)\frac{p\log_2 p}{n}}.\end{aligned} \tag{3.7}$$

Here, optimal processor number and highest attainable speedup are

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

and

$$S^{\text{opt}}_{\text{sws,cr}} = \frac{1}{4}p^{\text{opt}}_{\text{sws,cr}}\frac{1}{1+\log_2(p^{\text{opt}}_{\text{sws,cr}})},$$

respectively. As $p^{\text{opt}}_{\text{sws,cr}} = \mathcal{O}\left(\left(p^{\text{opt}}_{\text{sws}}\right)^2\right)$, $p^{\text{opt}}_{\text{sws,cr}}$ will grow much faster than $p^{\text{opt}}_{\text{sws}}$ for large problem sizes.

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 $4k^2$.

Remarks

This algorithm is well suited for *symmetric, positive definite* systems of equations, as the factorization of (3.2) is essentially symmetric. For symmetric matrices in (3.3) we have $F_i = E_i^T$ and $V_{i+1} = U_i^T$. The amount of work as well as the volume of the messages is reduced by a factor of two. Although the number of messages remains the same. Therefore, redundancy and speedup remain unchanged except for the weights for the startup time σ.

Wright [12] presented a version of the single width separator algorithm with pivoting. His algorithm incorporates complete pivoting. If the factorization of an A_i cannot be finished, 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.

Johnsson's analysis in [10] 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. In this analysis 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. The startup time includes the overhead due to initiation of a message transfer (system calls, buffer allocation). Thus, cyclic reduction is feasible on 2D grid connected processors (Intel Paragon).

4. Discussion

The singe width separator algorithm shares the same algorithmic structure with the other algorithms discussed in [2]. Two phases, factorization and back substitution, are naturally parallelizable. The third phase, the solution of the reduced system, is completely sequential. This phase forms the bottleneck for the algorithms.

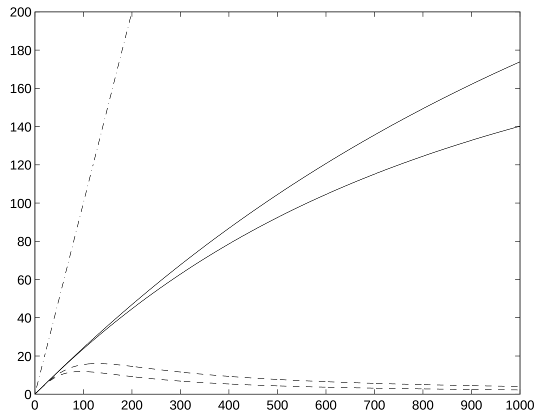


Figure 4.1: Theoretical speedups for $n = 100000$, $k = 10$, and $\tau = 1$. The pair of dashed lines corresponds to sws, the pair of solid lines to sws/cr. The upper and lower of the two lines correspond to $\sigma = 0$ and $\sigma = 1000$, respectively. The dash-dotted line indicates ideal speedup.

As the cost for the solution of the reduced system increases with the number of processors p, speedup will be maximal for some processor number p^{opt}. In addition, p^{opt} depends on the size of the problem. Clearly, these algorithms are not scalable. The variant of the algorithm incorporating cyclic reduction (sws/cr) is superior to the variant without (sws) since the denominator of the speedup formula of the latter contains a p^2 term instead of the $p\log_2(p)$ term of the former. By consequence, $p^{\text{opt}}_{\text{sws,cr}}$ is of the order of the square of $p^{\text{opt}}_{\text{sws}}$.

The *efficiency* of a parallel program is defined by

$$E(n,k,p) := \frac{S(n,k,p)}{p}. \tag{4.1}$$

Efficiency measures the fraction of the processor power that is actually utilized by the parallel program. It is evident that for fixed problem size the efficiency decreases if the processor number is increased. To maintain efficiency when increasing the number of processors, the problem size must also be increased. For our algorithms, the matrix size has to grow very rapidly to that end. From equations (3.6), (3.7), and (4.1) 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.

In Figure 4.1 the theoretical speedups according to formulae (3.6) and (3.7) are plotted for sws and sws/cr for $\sigma = 1000$ 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. The curves show that speedups for sws will be 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. Efficiency ranges between 10% and 20%. With sws/cr much higher speedups are obtained. However, the efficiency is still not high. Speedup and efficiency would be very satisfactory if they were given with respect to the performance of the *parallel* algorithm on one node as the high redundancy of this algorithm would then be neglected.

5. Numerical experiments

We ran the symmetric single width separator algorithm for 4 different problem sizes on the Intel Paragon at ETH Zurich [1]. 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 tests we always used the fast memory. Times for problems too large for memory were extrapolated from smaller problems. In this way we avoid performance losses due to page swapping. On the other hand, speedups are smaller than actually observed as overheads involved with solving large problems on a single processor are neglected. All times presented are the best obtained in several runs.

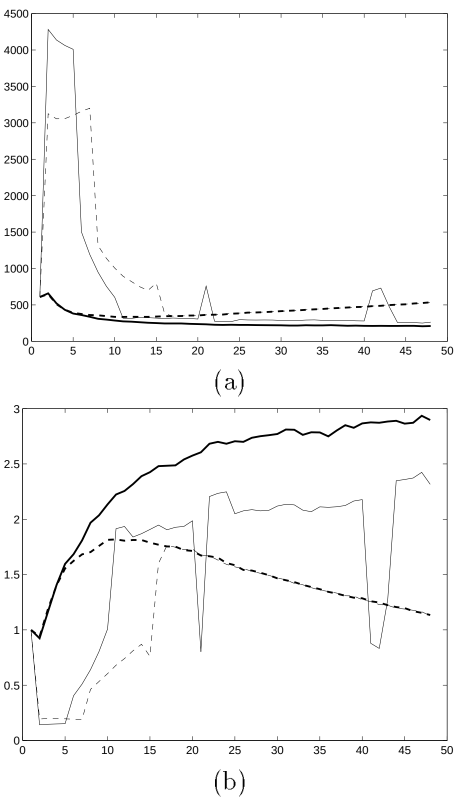


Figure 5.1: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 10000$ and $k = 10$. The thin/thick curves correspond to calculations with IEEE arithmetic turned on/off.

Figure 5.1 shows the times for the smallest problem size $(n,k) = (10000, 10)$ together with the corresponding speedups for machine partitions up to 48 nodes. The Fortran programs have been compiled with the IEEE flag turned on and off. It is observed that the cost of the floating point arithmetic according to the IEEE standard is very high. On the Intel Paragon, the IEEE division and square root are written in software. In addition the exception handling for the multiplication is done in software. In our numerically well conditioned examples there were only small differences in the accuracy of the results.

Timings and speedups for the problem size $(n,k) = (100000, 10)$ are given in Fig. 5.2. Again,

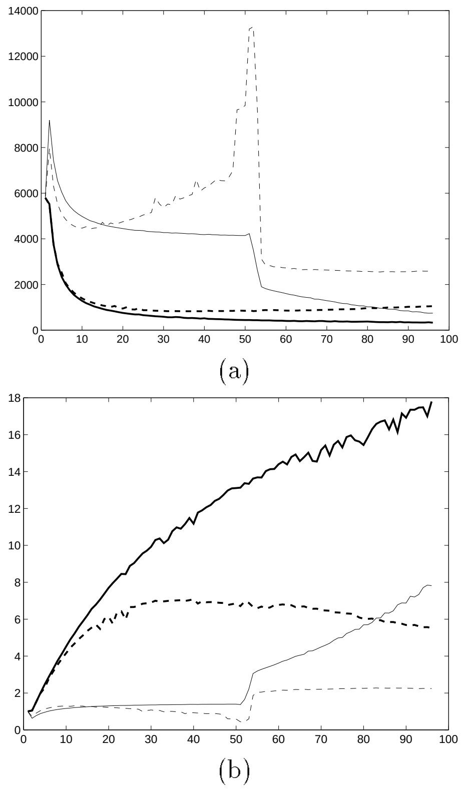


Figure 5.2: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 100000$ and $k = 10$.

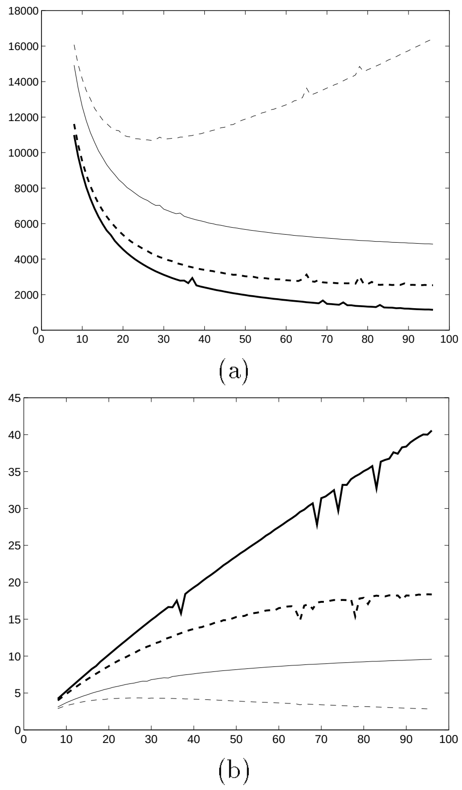


Figure 5.3: Times (a) and speedups (b) for sws (- -) and sws/cr (—) for $n = 800000$ and $k = 10$.

the timing curves obtained without IEEE arithmetic are much better than those with the IEEE flag turned on. The numbers obtained with IEEE arithmetic turned off qualitatively behave much like theory predicts, cf. Fig. 4.1. The actual speedups are slightly below the ones predicted. sws has its peak speedup at $p \approx 35$ while the speedup for sws/cr increases throughout the entire domain. Efficiencies range from 15% to 20%. The timings for the runs with IEEE arithmetic show a sudden drop at $p = 53$. The form of this performance 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 is not yet understood.

In Figures 5.3 and 5.4 times and speedups for the two big problem sizes $(n, k) = (800000, 10)$ and $(n, k) = (800000, 40)$ are plotted. The one processor times were obtained by linear extrapolation as neither of the problems fit into the memory of one processor. Leaving k fixed, we took the smallest n such that the problem fit into memory. Linear extrapolation was motivated by the linear dependence on n of the Cholesky factorization. The (800000,10) problem could be solved on one processor using the slow secondary memory in 600 seconds! Speedups with respect to this number greatly exceeded p. The $(800000, 40)$ problem was too large to solve on one processor even when we used the secondary memory. For both problem sizes, speedups are very satisfactory for the non-IEEE version of the programs, in particular for sws/cr. Because of the wider band, speedups are a bit smaller for $(n, k) = (800000, 40)$. The speedup curve for sws/cr is still increasing at the upper end of the plots. In the $(800000, 40)$ problem the IEEE versions were up to 30 times slower than the non-IEEE versions of the program! The difference were not as big with the smaller bandwidth as the part of the program that solves the reduced system and is written in generic Fortran consumes relatively less time. The rest of the programs are essentially calls to LAPACK subroutines which are optimized by Intel.

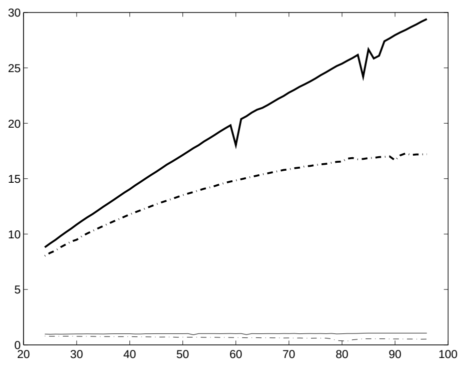


Figure 5.4: Times for sws (- -) and sws/cr (—) for $n = 800000$ and $k = 40$.

6. Conclusions

In conclusion, it can be said that direct methods for solving banded systems of equations are only 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 is a processor number $p^{\rm opt}$ for which speedup is highest. This number depends on the ratio n/k. The speedup curve is very flat at its peak. So, in most cases large processor numbers cannot be employed profitable.

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, because a single processor version, in any form, does not exist! It may be impossible (or excessively slow) to solve a large problem on a small number of processors because of its size.

Dongarra and Sameh [6] 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. However, 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] P. Arbenz. First experiences with the Intel Paragon. *SPEEDUP*, 8(2), 1994.

[2] P. Arbenz and W. Gander. A survey of direct parallel algorithms for banded linear systems. Tech. Report 221, ETH Zürich, Computer Science Department, November 1994.

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

[4] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36:177–195, 1981.

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

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

[7] K. Gates and P. Arbenz. Parallel divide and conquer algorithms for the symmetric tridiagonal eigenproblem. Tech. Report 222, ETH Zürich, Computer Science Department, November 1994. Submitted to SIAM J. Sci. Comput.

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

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

[11] D. C. Sorensen and P. T. P. Tang. On the orthogonality of eigenvectors computed by divide-and-conquer techniques. *SIAM J. Numer. Anal.*, 28:1752–1775, 1991.

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