ARBENZ, P.; GANDER, W.

Direct Parallel Algorithms for Banded Linear Systems

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

We consider the system of linear equations

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

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 (1). The cost for LU factorization, forward and backward substitution is

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

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

In [2] we surveyed the most important algorithms for solving (1) on distributed memory multiprocessor computers that support the MIMD programming model. In this note we discuss one of these algorithms, the so-called single width separator algorithm.

2. The single-width separator approach

This algorithm has been studied e.g. by Johnsson [7], Dongarra and Johnsson [4], and in modified forms by Wright [8] 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

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

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 $n_i > k$.

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) becomes

$$\left[\begin{array}{cccc|cccc} A_1 & & & & B_1 & & & \\ & A_2 & & & B_2 & B_3 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & A_{p-1} & & & & \ddots & B_{2p-3} \\ & & & A_p & & & & B_{2(p-1)} \\ \hline C_1 & C_2 & & & D_1 & & & \\ & C_3 & \ddots & & & D_2 & & \\ & & \ddots & \ddots & & & \ddots & \\ & & C_{2p-3} & C_{2(p-1)} & & & & D_{p-1} \end{array}\right] \left[\begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \\ \hline \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{array}\right] = \left[\begin{array}{c} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{p-1} \\ \mathbf{b}_p \\ \hline \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_{p-1} \end{array}\right]. \tag{4}$$

We denote the matrix in (4) by $\tilde{A}$. As matrices appearing in domain decomposition methods have a structure similar to $\tilde{A}$ the above permutation is sometimes called 'algebraic domain decomposition'.

The algorithm proceeds in three phases. In the *factorization phase*, we compute an incomplete block LU factorization [6] (one step of block cyclic reduction) of $\tilde{A}$, $\tilde{A} = LR$. After multiplying (4) by L^{-1}, we obtain the system

$$\left(\begin{array}{cccc|cccc} R_1 & & & & E_1 & & & \\ & R_2 & & & E_2 & E_3 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & R_{p-1} & & & & \ddots & E_{2p-3} \\ & & & R_p & & & & E_{2p-2} \\ \hline & & & & T_1 & U_1 & & \\ & & & & V_2 & T_2 & \ddots & \\ & & & & & \ddots & \ddots & U_{p-2} \\ & & & & & & V_{p-1} & T_{p-1} \end{array}\right) \left(\begin{array}{c} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{p-1} \\ \mathbf{x}_p \\ \hline \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-1} \end{array}\right) = \left(\begin{array}{c} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_{p-1} \\ \mathbf{c}_p \\ \hline \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_{p-1} \end{array}\right), \tag{5}$$

where $E_{2i-2} = L_i^{-1}B_{2i-2}$, $E_{2i-1} = L_i^{-1}B_{2i-1}$, $F_{2i-2} = C_{2i-2}R_i^{-1}$, $F_{2i-1} = C_{2i-1}R_i^{-1}$, $\mathbf{c}_i = L_i^{-1}\mathbf{b}_i$, $\boldsymbol{\gamma}_i = \boldsymbol{\beta}_i - F_{2i-1}\mathbf{c}_i - F_{2i}\mathbf{c}_{i+1}$, $T_i = D_i - F_{2i-1}E_{2i-1} - F_{2i}E_{2i}$, $U_i = -F_{2i}E_{2i+1}$, and $V_i = -F_{2i-1}E_{2i-2}$. 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*

$$\begin{bmatrix} T_1 & U_1 & & & \\ V_2 & T_2 & U_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & U_{p-2} \\ & & & V_{p-1} & T_{p-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \vdots \\ \boldsymbol{\xi}_{p-2} \\ \boldsymbol{\xi}_{p-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\gamma}_1 \\ \boldsymbol{\gamma}_2 \\ \vdots \\ \boldsymbol{\gamma}_{p-2} \\ \boldsymbol{\gamma}_{p-1} \end{bmatrix}. \tag{6}$$

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

In the second phase of the algorithm, the block tridiagonal diagonally dominant reduced system is solved by block Gaussian elimination or by block cyclic reduction. This phase requires communication. It is the bottleneck of the algorithm.

In the third phase, *back substitution*, the $\mathbf{x}_i$ are computed in parallel from the now known vectors $\boldsymbol{\xi}_{i-1}$ and $\boldsymbol{\xi}_i$.

$$\begin{aligned} \mathbf{x}_1 &= R_1^{-1}(\mathbf{c}_1 - E_1\boldsymbol{\xi}_1), \\ \mathbf{x}_i &= R_i^{-1}(\mathbf{c}_i - E_{2i-2}\boldsymbol{\xi}_{i-1} - E_{2i-1}\boldsymbol{\xi}_i), \ i \neq 1, p \\ \mathbf{x}_p &= R_p^{-1}(\mathbf{c}_p - E_{2p-2}\boldsymbol{\xi}_{p-1}). \end{aligned} \tag{7}$$

To discuss redundancy and speedup of the algorithm, we assume that $k := r = s$. If the reduced system is solved by block Gaussian elimination, the serial and parallel complexity of the single width separator algorithm is

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

and

$$C_{\rm sws}^{\rm par} \approx 8k^2\frac{n}{p} + \left(\frac{14}{3}k^3 + 2\sigma + k^2\tau\right) p \text{ flops}, \tag{9}$$

respectively. Here, 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. Solving the reduced system by block cyclic reduction lowers the cost to

$$C_{\rm sws,cr}(n,k,p) \approx \left(8k^2 - 6k^2\frac{1}{p}\right) n + \frac{14}{3}pk^3 \text{ flops} \tag{10}$$

and

$$C_{\rm sws,cr}^{\rm par} \approx 8k^2\frac{n}{p} + (\frac{14}{3}k^3 + 4\sigma + 4k^2\tau)\log_2 p \text{ flops}, \tag{11}$$

respectively. We note that, independent on how the reduced system is solved, the *redundancy* of the algorithm, i.e. the ratio of the serial complexities of the parallel and the optimal sequential algorithm, cf. (2), is about 4 for not too small numbers of processors. This bounds efficiency by 25%.

Speedup is the factor by which a parallel algorithm on p processors is faster than the best sequential algorithm. Depending on how the reduced system is solved, we have

$$S_{\rm sws}(n,k,p) \approx \frac{p}{4 + \left(\frac{7}{3}k + \frac{1}{2}\tau + \frac{\sigma}{k^2}\right)\frac{p^2}{n}} \quad \text{and} \quad S_{\rm sws,cr}(n,k,p) \approx \frac{p}{4 + \left(\frac{7}{3}k + 2\tau + 2\frac{\sigma}{k^2}\right)\frac{p\log_2 p}{n}}. \tag{12}$$

Clearly, the speedups are 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 if (2) 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 or again with increased redundancy. The p^2 or $p\log_2 p$ term in the speedup formula (12) cause speedup to *decrease* as soon as p exceeds a critical number $p^{\rm opt}$ for which speedup is highest. One easily deduces, that $p_{\rm sws}^{\rm opt} = \mathcal{O}(\sqrt{n/k})$ and $p_{\rm sws,cr}^{\rm opt} = \mathcal{O}(n/k)$

This algorithm is well suited for *symmetric, positive definite* systems of equations, as the factorization (5) is essentially symmetric. We have $R_i = L_i^T$, $F_i = E_i^T$, 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. Redundancy and speedup remain essentially unchanged [2].

Wright [8] 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. Johnsson's analysis in [7] 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 do 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).

3. Numerical experiments

We ran the symmetric single width separator algorithm 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. All speedups presented are the best obtained in several runs.

Figure 1(a) shows speedups for the smallest problem size $(n,k) = (10000, 10)$ 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

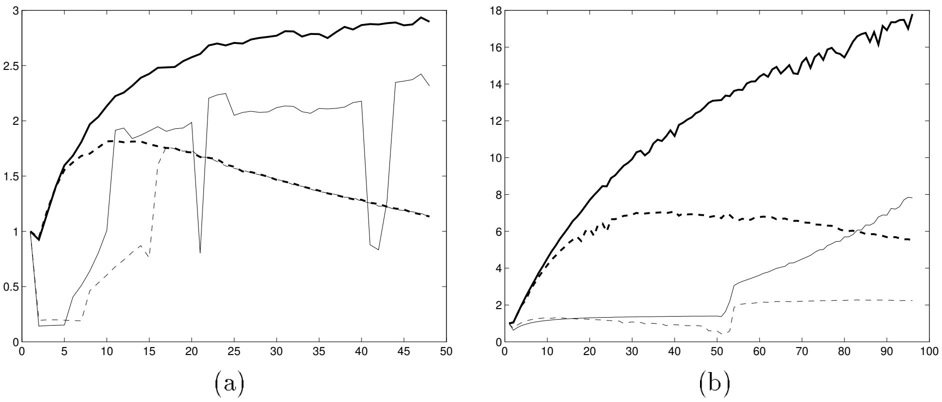

Figure 1: Speedups for sws (- -) and sws/cr (—) for (a) $n = 10000$ and $k = 10$ and for (b) $n = 100000$ and $k = 10$. The thin/thick curves correspond to calculations with IEEE arithmetic turned on/off.

floating point arithmetic according to the IEEE standard is very high. On the Intel Paragon, the IEEE division and square root as well as the exception handlings are done in software. In our numerically well conditioned examples there were only small differences in the accuracy of the results. Speedups for the problem size $(n, k) = (100000, 10)$ are given in Fig. 1(b). The numbers obtained with IEEE arithmetic turned off qualitatively behave much like theory predicts. The actual speedups are slightly below the ones predicted. For $(n, k) = (100000, 10)$, 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 erratic behavior of the IEEE versions are not understood yet.

4. 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^{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 [5] 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.

5. References

1 Arbenz, P.: First experiences with the Intel Paragon. SPEEDUP **8(2)** (1994), 53–60.

2 Arbenz, P.; Gander, W.: A survey of direct parallel algorithms for banded linear systems. Tech. Report 221, ETH Zürich, Computer Science Department, November 1994. `ftp://ftp.inf.ethz.ch/doc/tech-reports/1994/221.ps`.

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

4 Dongarra, J. J.; Johnsson, L.: Solving banded systems on a parallel processor. Parallel Computing **5** (1987), 219–246.

5 Dongarra, J. J.; Sameh, A. H.: On some parallel banded system solvers. Parallel Computing **1** (1984), 223–235.

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

7 Johnsson, S. L.: Solving narrow banded systems on ensemble architectures. ACM Trans. Math. Softw. **11** (1985), 271–288.

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

Addresses: Dr. Peter Arbenz, Prof. Dr. Walter Gander, Institute for Scientific Computing, Swiss Federal Institute of Technology (ETH), 8092 Zurich, Switzerland (`[arbenz,gander]@inf.ethz.ch`)