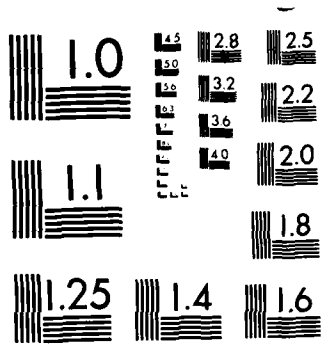
AD-A151 970 COMPLEXITY OF DENSE LINEAR SYSTEM SOLUTION ON A MULTIPROCESSOR RING(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE I C IPSEN ET AL. JAN 85 1/1

UNCLASSIFIED YALEU/DCS/RR-349 N00014-82-K-0184 F/G 12/1 NL

END

FILMED

DTIC



YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract. Different algorithms, based on Gaussian elimination, for the solution of dense linear systems of equations, are discussed for a multiprocessor ring. The number of processors is assumed not to exceed the problem size. A fairly general model for data transfer is proposed and the algorithms are analysed with respect to their requirements of arithmetic as well as communication times.

Complexity of Dense Linear System Solution on a Multiprocessor Ring

Ilse C.F. Ipsen, Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-349
January 1985

This work was supported in part by by ONR grant N00014-82-K-0184 and in part by a joint study with IBM/Kingston

1. Introduction

This paper discusses various algorithms, based on Gaussian Elimination, for the solution of *dense* linear systems of equations,

$$Ax = b,$$

on a linearly connected ring of general purpose processors.

In multiprocessor systems, the total time to perform a sequence of computational tasks does not only depend on *when* a task is completed but also *where* (i.e. in which processor) it is accomplished. This in turn implies a great richness in the class of algorithms, in terms of the assignments of tasks to processors and the assumed topology of the processor communication network. For a particular task it is now important *in which* processor its input data are situated (ie, how long it takes to move them to the requesting processor) and *when* they are available.

As the number of processors will not exceed the problem size, and will usually be much smaller, the algorithms differ in the way the matrix A is distributed among the processors.

1.1. Overview

The approach taken here for the development and analysis of algorithms acknowledges that times for data transfer communication are *not* negligible and may in fact dominate the times for actual arithmetic. A fairly general communication model is proposed, and all algorithms are characterised and compared with respect to their requirements for arithmetic as well as communication.

Following the classical approach, methods for triangular system solution are discussed (section 3) before introducing schemes for the Gaussian elimination (section 4). To begin with, however, the second part of this section presents a summary of the requisite hardware features, based on which various ways of transferring data can be devised (section 2).

To avoid long, non-descriptive formulae in favor of simpler results, the derivation of arithmetic and communication times will contain merely high order terms (in the problem size and the processor count). Furthermore, only a few representative methods will be described in detail to illustrate their analysis, while other, obvious variations, will be listed in tables.

Surveys of (general) parallel algorithms for the direct solution of dense linear systems of equations appear in [3, 9, 10]. Probably the earliest paper to realise that 'data movement, rather than arithmetic operations, can be the limiting factor in the performance of parallel computers on matrix operations' is [2]. There, lower bounds for matrix multiplication and matrix inversion are determined for arbitrary processor interconnection schemes where each processor can hold one matrix element. In [1], the communication requirements of some numerical methods, such as tridiagonal system solution by substructuring, ADI, FFT and fast Poisson solvers, are analysed with respect to shared-memory multiprocessors and highly parallel non-shared-memory MIMD systems. A probabilistic model for predicting iteration time and optimal data allocation when solving linear systems via iterative methods is presented in [6]. Its application in [7] prompts the conclusion that 'a broadcast bus architecture *can* effectively reduce the expected computation time for solving sparse linear systems.'

Gaussian elimination for dense systems on a multiprocessor ring is discussed in [8] Lawrie and Sameh [4] present a technique for solving symmetric positive definite banded systems, which is a generalization of a method for tridiagonal system solution on multiprocessors; it takes advantage of different alignment networks for allocating data to the memories of particular processors. However, the analysis in both is based on the assumption that the time for transmitting one floating point number from a processor to its nearest neighbor does not exceed the time for an arithmetic operation.

This paper lays no claims to being either exhaustive or complete. Its objective is to compare a variety of algorithms, which are fairly reasonable to program and to analyse, for the solution

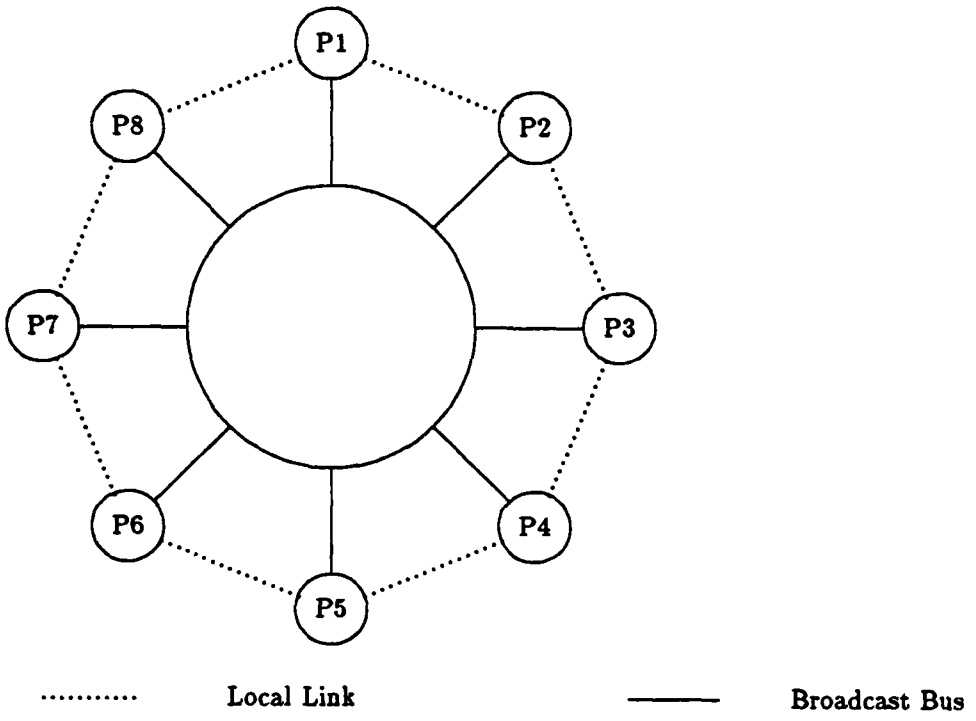


Figure 1: Ring of $k = 8$ Processors.

of a single problem on a certain class of parallel architectures, thereby leading to a more realistic approach to future algorithm development on multiprocessor machines.

1.2. The Multiprocessor Ring

The multiprocessor architecture under consideration, depicted in Figure 1, was introduced in [11] and consists of

- a 'ring' of a small number of k linearly connected general purpose (possibly pipelined) processors, each with its own memory (the processors in the ring will be consecutively numbered P_1 through P_k),
- a fast bus used mainly for broadcasting or transferring data at high speed,
- local interconnections linking each processor to its nearest neighbors (the ring).

It is assumed, that any processor is capable of writing to one neighbor while reading from the other, using the local links. For purposes of estimating the computation time, processors are considered to work in lock step where one step corresponds to the computation time of the slowest processor.

In order to discern the merits and disadvantages of the local links on one hand and the broadcast bus on the other, they will be used separately and analysed one at a time.

Assume the bus has a speed of R_B words per second while the local links can transfer data at a rate of R_L words per second. The inverses of R_B and R_L are denoted by τ_B and τ_L respectively. To be general, each transfer of a data *packet* is associated with a *constant* start-up (set-up) time of β_B and β_L, respectively, which is independent of the size (the number of words) per packet. Often, the start-up times are (much) larger than the elemental transfer times, that is,

$$\beta_B \gg \tau_B, \quad \beta_L \gg \tau_L.$$

The time to broadcast a packet of size N via the bus is

$$t_{T,B} = \beta_B + N\tau_B,$$

while the time to send it from a processor to its neighbor by using the local links is

$$t_{T,L} = \beta_L + N\tau_L.$$

On a single processor, a linear combination of two vectors of length N takes time

$$t_A = \gamma + N\omega,$$

where γ is the pipe fill time (it is zero for non-pipelined machines), ω the time for one scalar operation and $\gamma \geq \omega$ (again, the start-up time dominates the elemental operation time). In this paper, $t_{T,B}$ denotes data transfers times for the bus, $t_{T,L}$ refers to those involving the local links and t_A stands for arithmetic times. For any algorithm the sum of its transfer and arithmetic time, $t_{T,*} + t_A$, is simply called its *computation time*.

Whenever convenient, we assume without loss of generality that the problem size N is a multiple of the number of processors, k.

2. Data Transfers

In this section we consider different ways of transferring data among processors which are important in subsequent computational algorithms. We assume that a vector can be divided up into 'packets' of arbitrary size (subject to the vector length, of course).

As mentioned in the previous section, it takes time

$$t_{T,B} = \beta_B + N\tau_B$$

to broadcast a vector of length N from one processor to all others using the broadcast bus. Consequently, the time to broadcast a vector is independent of the number of destination processors.

An alternative method consists of using only the local links between the processors and pipelining the data transfers : while sending one packet to its successor a processor receives the next packet from its predecessor. Thus, if processor P_1 is to send its data to all other processors, then in step 1 the first packet is sent from P_1 to P_2. In step j, the first packet is sent from P_j to P_{j+1} while the second packet follows up from P_{j-1} to P_j, etc.

If the vector is partitioned into ν packets of equal size, then the process will terminate after $k + \nu - 2$ steps, when the last packet has reached P_k. With regard to high order terms in k and ν, the total time comes to

$$t_{T,L} \approx (k + \nu)\beta_L + (k + \nu)\frac{N}{\nu}\tau_L. \tag{2.1}$$

The above equation indicates that for large enough ν the time required for transferring a vector of length N is proportional to $N\tau_L$. However, the larger ν, the larger the cost of the set-up times will be.

Another possibility is to have P_1 send its data 'both ways round' so that processors to the left and right of P_1 would receive them at the same time. The according data transfer time comes to

$$t_{T,L} \approx (\frac{1}{2}k + \nu)\beta_L + (\frac{1}{2}k + \nu)\frac{N}{\nu}\tau_L,$$

which is at most twice as fast as the one in (2.1) when data are sent to all k processors. If fewer than k processors are to receive data, this scheme becomes more complicated and vectors might

have to be partitioned into packets of differing size. Since the difference is only a factor of two, we prefer to restrict ourselves to the simpler scheme of 'one way data flow.'

From equation (2.1) one observes that an optimal value for ν exists and is given by

$$\nu_{opt} = \sqrt{kN\frac{\tau_L}{\beta_L}} \tag{2.2}$$

for which the optimal time becomes

$$\begin{aligned} t_{T,L,opt}(N) &\approx N\tau_L + k\beta_L + 2\sqrt{kN\tau_L\beta_L} \\ &= \left(\sqrt{N\tau_L} + \sqrt{k\beta_L}\right)^2. \end{aligned} \tag{2.3}$$

Observe, that in practice $1 \leq \nu \leq N$, so that formula (2.2) is valid only when

$$1 \leq \frac{N}{k}\frac{\beta_L}{\tau_L} \leq N^2.$$

Otherwise, the optimal time simply becomes

$$t_{T,L,opt}(N) \approx (k+N)(\beta_L + \tau_L), \qquad \text{if} \quad \frac{N}{k}\frac{\beta_L}{\tau_L} < 1,$$

and

$$t_{T,L,opt}(N) \approx k(\beta_L + N\tau_L), \qquad \text{if} \quad \frac{N}{k}\frac{\beta_L}{\tau_L} > N^2.$$

For example, assume that k processors with transmission time τ_L and a problem of size N are given. If, with increasing set-up time for data transmission, the transfer time is to remain optimal, the number of packets must decrease (while their size increases), so that a smaller number of set-up times is required. Yet, if elemental transmission and set-up time are of the same order of magnitude, then the packets should each be of size $\sqrt{\frac{N}{k}}$. Sometimes we will make use of the double inequality

$$N\tau_L + k\beta_L \leq t_{T,L,opt}(N) \leq 2(N\tau_L + k\beta_L). \tag{2.4}$$

Note that the upper bound corresponds to choosing the non-optimal value $\nu = k$.

Sending a vector to processors that are not more than a distance of $\tilde{k} < k$ away changes the optimal value of ν to

$$\nu_{opt} = \sqrt{\tilde{k}N\frac{\tau_L}{\beta_L}}$$

and the corresponding time to

$$t_{T,L,opt}(N) \approx \left(\sqrt{N\tau_L} + \sqrt{\tilde{k}\beta_L}\right)^2. \tag{2.5}$$

Obviously, it does not pay to divide a vector into packets when using the broadcast bus.

In the case where a vector v is 'uniformly' distributed over the k processors so that the subvector v_i of length $\frac{N}{k}$ resides in processor P_i, an important operation is $\oplus v_i$, the direct sum of the blocks $v_1 \ldots v_k$, which makes the full vector v available in each processor. This obviously requires no computation but only data transfers.

Using the bus, it is possible to broadcast each v_i, one after the other, to all the processors which requires time

$$t_{T,B} = k\beta_B + k(\frac{N}{k})\tau_B = k\beta_B + N\tau_B.$$

When employing the local links, the subvectors are 'rotated' in a roundabout fashion : in step 1 we simultaneously send v_1 from P_1 to P_2, v_2 from P_2 to P_3, etc, and finally v_k from P_k to P_1; generally, in step j, we transmit v_1 from P_j to P_{j+1}, v_2 from P_{j+1} to P_{j+2}, and v_k from P_{j-1} to P_j (the indices should be taken modulo k). After k of the above steps v_i has encountered each processor. Hence the whole process requires time

$$t_{T,L} = k\beta_L + k(\frac{N}{k})\tau_L = k\beta_L + N\tau_L. \tag{2.6}$$

Consequently, the number of set-up times and elemental transfer times is the same for bus and local links.

In the algorithms for solution of dense linear systems, not much difference will be apparent in computation times involving broadcasting or pipelined data transfer. Even though they differ in the number of start-up times, these times (for broadcasting as well as local links) occur only with *low order terms* of the problem size, N, and hence have asymptotically no influence on the high order terms which are relevant for the overall time estimate. However, the final judgement can be made only when benchmarks from real multiprocessor systems are available.

3. Solution of Triangular Systems

In sequential machines, a general dense linear system

$$Ax = b$$

is efficiently solved by first reducing it to triangular form by Gaussian elimination and then solving the resulting triangular system. The same approach will be used for a parallel implementation on the multiprocessor ring. As is classically done, we will start by considering the solution of triangular systems.

3.1. Partitioning the Matrix into Blocks of Contiguous Rows

Consider the upper triangular system

$$Ux = b, \tag{3.1}$$

where U is an upper triangular matrix of size $N \times N$. The simplest idea that comes to mind for the solution of such a system on a parallel machine, is to partition the rows of U into k blocks each consisting of N/k rows and to store each block in a processor, as shown in Figure 2. Recall that the processors are numbered consecutively $P_1, P_2, ..P_k$.

Let processor P_i hold rows $(i-1)\frac{N}{k}+1$ to $i\frac{N}{k}$ of U, the corresponding block b_i of the right hand side vector b, and the block x_i of the solution vector x. Accordingly, denote by U_{ij} the $N/k \times N/k$ *block* matrix in position (i,j) of the matrix U. The algorithm TRB (Triangular system solution with Block Rows) to solve (3.1) is shown below; proceeding from bottom to the top of the matrix (i.e., $i = k, k-1, \ldots 1$), processor P_i solves the $\frac{N}{k} \times \frac{N}{k}$ triangular system with the ith diagonal block as coefficient matrix. The solution vector x_i is then sent to the processors to the left of P_i which perform the corresponding matrix-vector multiplications with x_i.

Figure 3 graphically illustrates the algorithm, entries in the matrix U denote the time steps when the corresponding block matrix U_{ij} is processed.

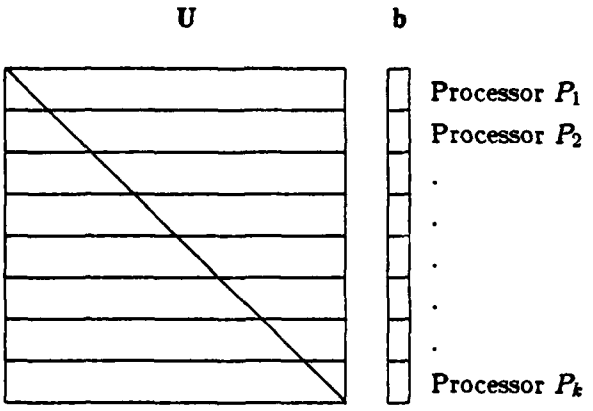


Figure 2: Block-Row Partitioning of a Triangular System.

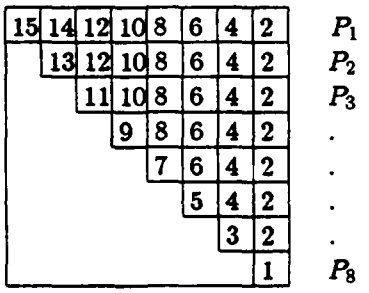


Figure 3: Sketch of Algorithm TRB for $k = 8$.

ALGORITHM TRB (Triangular system solution with Block Rows)

1. Solve in P_k

$$U_{kk}x_k = b_k$$

2. For $i = k-1, k-2, \ldots 1$ do
   (a) Send x_{i+1} from P_{i+1} to $P_i, P_{i-1}, \ldots P_1$
   (b) For $j = 1, 2, \ldots i$ do in P_j

$$b_j := b_j - U_{j,i+1}x_{i+1} \tag{3.2}$$

   (c) Solve in P_i

$$U_{ii}x_i = b_i. \tag{3.3}$$

At each step i of Algorithm TRB a vector of length N/k must be transferred from P_{i+1} to $P_i, P_{i-1}, \ldots P_1$ which, according to (2.5), requires time

$$\frac{N}{k}\tau_L + i\beta_L + 2\sqrt{\frac{N}{k}i\tau_L\beta_L} = \left(\sqrt{\frac{N}{k}\tau_L} + \sqrt{i\beta_L}\right)^2 \leq 2(\frac{N}{k}\tau_L + i\beta_L) \tag{3.4}$$

using the local links and

$$\beta_B + \frac{N}{k}\tau_B \tag{3.5}$$

using the broadcasting bus. Summing up over steps $i = k-1, k-2, \ldots 1$ yields the total times required for data communication

$$\begin{aligned} t_{T,L} &\approx N\tau_L + \frac{k^2}{2}\beta_L + \frac{4}{3}\sqrt{\frac{N}{k}\tau_L\beta_L}k^{3/2} = N\tau_L + \frac{k^2}{2}\beta_L + \frac{4}{3}k\sqrt{N\tau_L\beta_L k} \\ &\leq 2N\tau_L + k^2\beta_L, \end{aligned} \tag{3.6}$$

and

$$t_{T,B} \approx N\tau_B + k\beta_B, \tag{3.7}$$

where the approximations

$$\sum_{i=1}^{k-1} i \approx \frac{k^2}{2}, \qquad \sum_{i=1}^{k-1}\sqrt{i} \approx \frac{2}{3}k^{3/2} \tag{3.8}$$

have been employed, which are valid only when k is sufficiently large.

Now consider the time spent doing arithmetic. At each step in (3.2) a matrix-vector multiplication involving

$$\frac{N}{k}(\gamma + \frac{N}{k}\omega) \tag{3.9}$$

steps is performed in each active processor. The cost of solving the triangular system in (3.3),

$$Rx = f,$$

on a pipeline machine requires time

$$\frac{1}{2}\left(\frac{N}{k}\right)^2 \omega + \frac{N}{k}\gamma, \tag{3.10}$$

because x_i, $1 \leq i < \frac{N}{k}$, is obtained by

$$x_i = \frac{1}{r_{ii}}\left(f_i - \sum_{j=i+1}^{N/k} r_{ij}x_j\right)$$

and

$$x_{N/k} = f_{N/k}/r_{N/k},$$

where the indices are relative to the sub-block. The term in brackets constitutes an inner-product and can be performed in time $(\frac{N}{k} - i - 1)\omega + \gamma$. The division is incorporated into the pipelining of the inner-product so that there is no need for an additional start-up time. However, for some machines (the FPS-164, for instance) this may have to be revised as divsions are significantly more expensive than additions or multiplications. Summing (3.9) and (3.10) over $k - 1$ steps gives an arithmetic time of

$$t_A \approx \frac{3}{2}\frac{N^2}{k}\omega + 2N\gamma. \tag{3.11}$$

3.2. Partitioning the Matrix into Blocks of Contiguous Columns

In a block column-oriented partitioning scheme each of the k processors contains $\frac{N}{k}$ adjacent columns, that is, processor P_i contains columns $(i-1)\frac{N}{k} + 1$ to $i\frac{N}{k}$ of U, as well as b_i, where U_{ij} is again the $\frac{N}{k} \times \frac{N}{k}$ block matrix at position (i,j) of U. Although this algorithm will turn out to be the most inefficient one presented in this paper, it is described on account of its simplicity and in order to better illustrate related improved versions.

During each step, a triangular subsystem of order $\frac{N}{k} \times \frac{N}{k}$ is solved, whereafter all matrix vector products $y_{ij} = U_{ij}x_j$ for the next higher block row are performed *in parallel* and the partial sums y_{ij} are sent to the processor responsible for the subsequent triangular system solution. Algorithm TCB (Triangular system solution with Block Columns) is graphically sketched in Figure 4 where the entry for U_{ij} contains the time step at which it participates in a computation. The algorithm is formulated for data transfers involving the local links, the modifications for the bus are obvious.

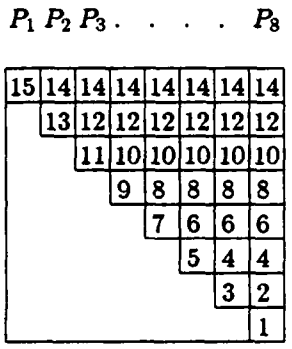


Figure 4: Sketch of Algorithm TCB for $k = 8$.

ALGORITHM TCB (Triangular system solution with Block Columns)

1. Solve in P_k

$$U_{kk}x_k = b_k$$

2. For $i = k-1, k-2, \ldots 1$ do
   (a) For $j = k, k-1, \ldots i+1$ do in P_j

$$y_{ij} := U_{ij}x_j \tag{3.12}$$

   (b) For $j = k, k-1, \ldots i+1$ do in P_j
   Send y_{ij} to P_{j-1}
   (c) Solve in P_i

$$U_{ii}x_i = b_i - \sum_{j=i+1}^{k} y_{ij}. \tag{3.13}$$

Observe that the solution vector parts x_i are never transmitted, only the matrix-vector products y_{ij}. The communication time with the local links for 2(b) in step i comes to

$$(k-i-1)(\beta_L + \frac{N}{k}\tau_L)$$

since all y_{ij} can be sent at the same time via the local links. For $k-1$ steps this makes

$$t_{T,L} \approx \frac{1}{2}k^2\beta_L + \frac{1}{2}kN\tau_L. \tag{3.14}$$

Because ***different*** vectors y_{ij} must be sent to ***one*** processor P_i at the same time, broadcasting bears no advantage over the local links and

$$t_{T,B} \approx \frac{1}{2}k^2\beta_B + \frac{1}{2}kN\tau_B.$$

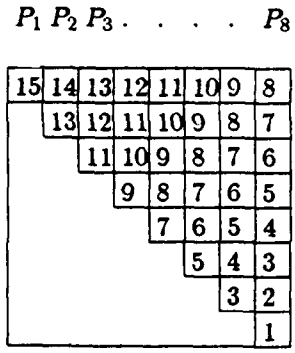


Figure 5: Sketch of Algorithm TCBG for $k = 8$.

The arithmetic time can be easily determined by observing that in each step the matrix-vector multiplications which are all done in parallel are *followed* by a triangular system solution. Hence, from (3.9) and (3.10) the time per step is

$$2\frac{N}{k}\gamma + \frac{3}{2}(\frac{N}{k})^2\omega,$$

and for k steps it is given by (3.11). The time for the vector additions in 2(c) is

$$\frac{1}{2}k^2(\gamma + \frac{N}{k}\omega),$$

and hence,

$$t_A = (2N + \frac{1}{2}k^2)\gamma + (\frac{3}{2}\frac{N^2}{k} + \frac{N}{k})\omega.$$

However, compared to method TRB, the communication time of TCB is worse by a factor of k. The reason being that the $k - i$ matrix vector multiplications in (3.12) of step i are all executed in parallel, and completed at the same time. Consequently, in step 2(b) $k - i - 1$ vectors are sent to *one* processor, which can only receive them in sequence.

It would seem that the computation time could be improved by performing the vector summations of step 2(c) in a 'logarithmic' fashion, since more computations and transfers could be done simultaneously. In that case, however, the distances to the destination processors would increase, to as much as $\frac{1}{2}k$ for the last summation. The fastest way might be, for a given i, to compute the first f_i, $f_i \leq \log(k - i)$, summations in logarithmic fashion and then send the partial sums and the remaining $\log(k - i) - f_i$ vectors to *one* processor for the computation of the final y_i.

3.3. Partitioning the Matrix into Blocks of Contiguous Columns : A Second Approach

Note that in Algorithm TCB *all* processors are waiting for the diagonal block-system, which is triangular, to be solved, so that all matrix vector products of *one* block row can be computed in parallel. However, in this second version of the column oriented method, TCBG (Greedy Triangular system solution with Block Columns), which might be regarded as a 'greedy method', each processor performs its matrix vector multiplications *as soon as possible*, see Figure 5. Again, processor P_i comprises columns $(i-1)\frac{N}{k}$ to $i\frac{N}{k}$ of U.

4.6. Gauss-Jordan Elimination

The Gauss-Jordan algorithm is one of the simplest approaches toward attempting to improve the arithmetic efficiency of Gaussian elimination on multprocessors. We consider a system partitioned in k blocks of m contiguous rows as shown in Figure 8. As was observed earlier, in the Gaussian elimination algorithm proposed in Section 4.1, we could use the idle processors to continue the elimination on the rows above the current pivot row thus maximizing the number of active processors, at any given step.

To estimate the transfer time, we observe that at each step we must send the pivot row to all processors. This results in a time identical with that of row scattered Gaussian elimination, i.e. given by (4.5). Similarly, the arithmetic time is identical with that of Gaussian elimination with block row partitioning, i.e. it is given by (4.4).

The obvious advantage of this method over that described in Section 4.1, is that we no longer have to solve a triangular system. Clearly, scattering of the matrix across the processors will not result in any gain here because all processors are busy during the whole elimination .

5. Partial Pivoting

So far, the issue of pivoting has been put aside in order to simplify the description of our algorithms. In fact as we now show, partial pivoting can be incorporated, at little extra cost.

First consider the block row method described in Section 4.1. At the j^{th} step, we need to search for the element of largest absolute value, among the elements $a_{ij}, i = j, ..N$. This can be achieved in two stages. In the first stage the maximum element is found in each processor. We refer to these as the local maxima. Then a comparison must be done between these local maxima to obtain the global maximum. The first step costs $\frac{N}{k}\omega'$ where ω' is the time for doing one comparison within any processor. For the second stage, using nearest neighbor connections, a round robin type comparison between the local maxima requires a communication time of at most $i(\tau_L + \beta_L)$ where i is the number of processors involved in the j^{th} step of Gaussian elimination , i.e. $i = k - \lceil j/m \rceil$. The same number of comparisons is also needed and this requires time $i\omega'$. A similar approach using the broadcast bus is to broadcast each local maximum in turn to the $i - 1$ other processors and do the comparisons for the global maximum in each of the i processors, in parallel. This would lead to similar times for communication and for identical times for arithmetic. Summing up the above over the $N - 1$ steps of the Gaussian elimination algorithm, one finds that the overhead for partial pivoting in the block-row scheme is

$$t_{P,BR} \approx \left(\frac{N^2}{k} + \frac{N}{2}k\right)\omega' + \frac{Nk}{2}(\tau + \beta), \tag{5.1}$$

where τ and β represent either τ_L and β_L, or τ_B and β_B depending on which, the local links or the bus, is used.

For the block-column scheme, all of the j^{th} column resides in one processor, so there is no need for transferring data. However, all comparisons are done in one processor while the others are idle. It is therefore clear that the resulting overhead time is

$$t_{P,BC} \approx \frac{N^2}{2}\omega'. \tag{5.2}$$

The scattered scheme of Section 3.5 is similar to that of the block row scheme, except that the comparisons of the second stage now take place among all k processors instead of only $i = k - \lceil j/m \rceil$ as above. As a result the overhead time for pivoting is double that of TRB, i.e.

$$t_{P,BRS} \approx \left(\frac{N^2}{2k} + Nk\right)\omega' + Nk(\tau + \beta). \tag{5.3}$$

1	2	3	4	5	6	7	8
8	1	2	3	4	5	6	7
7	8	1	2	3	4	5	6
6	7	8	1	2	3	4	5
5	6	7	8	1	2	3	4
4	5	6	7	8	1	2	3
3	4	5	6	7	8	1	2
2	3	4	5	6	7	8	1

Figure 10: Block Diagonal Scattering of a Linear System Across Eight Processors.

Note that the preceeding time is twice that of most other distributions, because data must be sent in two directions, West-East and North-South.

Using the broadcast bus, the only difference is that each piece of length m can now be moved simultaneously to all processors at the cost of $m\tau_B + \beta_B$. The resulting total transfer time is

$$t_{T,B} \approx N^2\tau_B + Nk\beta_B.$$

We consider now the arithmetic complexity of this method. In step j of the algorithm we perform $N - j$ eliminations each of which is split into several linear combinations of vectors of length m taking place in a different processor. The processor most delayed in performing these linear combinations is P_1 which holds the diagonal blocks. Processor P_1 performs exactly $N - j$ linear combinations of length m each. Hence the arithmetic time at step j is

$$(N - j)(m\omega + \gamma).$$

and the total arithmetic time is

$$t_A \approx \frac{N^3}{2k}\omega + \frac{N^2}{2}\gamma. \tag{4.8}$$

We note that it is possible to extend this scheme by considering blocks of size mxm with $1 \leq m \leq N/k$ and scattering them by diagonals, i.e. by assigning all the blocks on the same diagonal cyclically to processors $P_1, P_2, ..P_k, P_1, P_2, ...P_k, ...$ The purpose of this scattering is to improve efficiency by chosing the parameter m so that the total computation time is minimized. However, this will not be considered as it leads to complicated formulas and does not result in a significantly better algorithm.

Using the approximation

$$\left\lceil \frac{N-j}{k} \right\rceil \approx \frac{(N-j)}{k} \tag{4.6}$$

and summing over j, we have

$$t_A \approx \frac{1}{3}\frac{N^3}{k}\omega + \frac{1}{2}\frac{N^2}{k}\gamma. \tag{4.7}$$

Observe that as before the start-up time is reduced by a factor of k. Also note that the above formula is only valid for $k << N$, because approximation (4.6) was used.

Scattering the ***columns*** of the matrix across the processors does not change the communication time (4.5). Analogous to (4.7), the arithmetic time is

$$t_A \approx \frac{N^3}{3k}\omega + \frac{N^2}{2}\gamma,$$

where the start-up times are independent of k.

The communication time for scattered partitioning is always larger than that of the 'contiguous' partitioning of section 4.1. For large α, it is roughly twice as big. When α is small, the two times are comparable. Furthermore, scattering makes the arithmetic computation time consistent with the sequential case, ie, for $k = 1$ the arithmetic time reduces to that of the sequential evaluation.

4.4. Reduction to a DDB scattered matrix

As was mentioned in Sections 3.4 and 3.6, it is important to have diagonal blocks that are diagonal matrices. Such matrices can be obtained in the same time as regular upper triangular systems by simply taking advantage of the idle time of the processors in the regular versions. The result is a triangular system whose solution requires less communication and start-up times, see Section 3.

4.5. Partitioning the Matrix into Block Diagonals

Now consider a scheme which leads to the diagonal scattering of Section 3.7 by partitioning the linear system into square blocks of size $m \times m$ each, where $m = N/k$. Again, A_{ij} represents the block matrix in position (i,j) of A. The data are scattered so that block $A_{i,j}$ belongs to processor number $1 + [(i-j) \ mod \ k]$, $1 \le i,j \le k$. The above distribution is illutrated in Figure 10, where a matrix entry denotes the processor to which the corresponding block matrix is assigned. The right hand side b can be considered as an additional column of A and is distributed accordingly among the processors.

The motivation for considering such distributions, is the ease with which local link-data transfers can be overlapped, since any two contiguous blocks of A belong to neighboring processors.

At each step j of Gaussian elimination, the multipliers must be transferred downward. All processors holding that row will simultaneously communicate to those beneath them one piece of size $m = N/k$ of that row. This is possible because of the way the matrix is distributed. Only transfers from processor P_i to processor $P_{[(i-1) \ mod \ k]}$ are necessary. These transfers are repeated until each piece reaches the processor which holds the corresponding piece of the last block row. This requires exactly $\lfloor (N-j)/m \rfloor$ such transfer steps. Each of these steps consists of sending in parallel from one processor to a direct neighbor a vector of length N/k. Hence the time for communicating the j^{th} row is approximately

$$\frac{N-j}{m}(m\tau_L + \beta_L) = (N-j)\tau_L + \frac{N-j}{m}\beta_L.$$

Once the pivot row is available in all processors, the pivots also need to be transmitted, i.e., the j^{th} column to the right. Clearly, this involves the same amount of time as above. All this results in a total transfer time of

$$t_{T,L} \approx N^2\tau_L + Nk\beta_L.$$

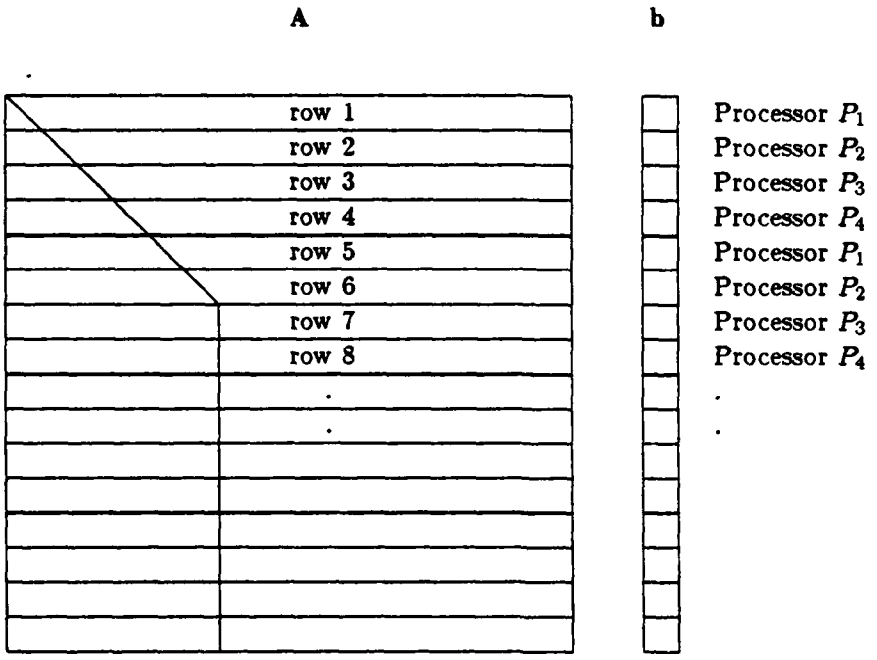


Figure 9: Gaussian elimination with Scattered Rows for $k = 4$.

4.3. Scattering the Rows and Columns of a Matrix

As in Section 3.5, the matrix A is partitioned into blocks of k rows or columns. However, now the processors do not contain blocks of contiguous rows or columns, but the rows (or columns) are scattered 'cyclically' across processors $P_1 \ldots P_k$. This scheme will be referred to as 'scattered Gaussian elimination.'

The scattering of rows is depicted in Figure 9. At step j, the pivot row must be available to all processors for purposes of elimination . Sending a row of length $N - j$ to all processors (see Section 2) takes time

$$(N-j)\tau_L + k\beta_L + 2\sqrt{k\beta_L\tau_L}\sqrt{N-j}.$$

Summing this expression for $j = 1, 2, \ldots N-1$ and using the approximations (3.8), we find an approximate total communication time of

$$t_{T,L} \approx \frac{N^2}{2}\tau_L + kN\beta_L + 2\sqrt{k\beta_L\tau_L}\frac{2}{3}\left(N^{3/2}\right) = \frac{N^2}{2}\tau_L\left(1 + \frac{8}{3}\alpha + 2\alpha^2\right), \tag{4.5}$$

where α is defined by (4.3).

As for arithmetic, there are $N - j$ elements to be eliminated during step j, and since the rows are scattered across the k processors, each processor will perform about $\lceil (N-j)/k \rceil$ linear combinations at the cost of $\gamma + (N-j)\omega$ each. Therefore, step j consumes time

$$\left\lceil \frac{N-j}{k} \right\rceil [\gamma + (N-j)\omega].$$

which can be written as

$$t_{T,L} \approx \frac{N^2}{2}\tau_L (1+\alpha)^2, \tag{4.2}$$

with

$$\alpha = \sqrt{\frac{k}{N}\frac{\beta_L}{\tau_L}}. \tag{4.3}$$

Using the bus to broadcast a row of length $(N-j)$ to an arbitrary number of processors requires the time

$$\beta_B + (N-j)\tau_B$$

and therefore the total communication time when using the bus is given by

$$t_{T,B} = N\beta_B + \frac{1}{2}N^2\tau_B.$$

To determine the time for arithmetic, we note that at step j a processor performs at most $\frac{N}{k}$ elimination s which takes time

$$\frac{N}{k}(\gamma + (N-j)\omega).$$

Summing up over $N-1$ steps,

$$t_A \approx \frac{N^3}{2k}\omega + \frac{N^2}{k}\gamma. \tag{4.4}$$

The reduction of the start-up time $N^2\gamma$ by the factor k in the above formula, comes from the fact that the active processors simultaneously eliminate (at most) $\frac{N}{k}$ elements per column by computing linear combinations of rows of length $N-j$.

4.2. Partitioning of the Matrix into Blocks of Contiguous Columns

Similarly, if the matrix is divided up into blocks of contiguous columns, then P_i contains columns $(i-1)\frac{N}{k}$ to $i\frac{N}{k}$ of A. For simplicity, the vector b is considered to be another column of A and hence stored in P_k. At step j, column j, which contains the multipliers is in P_i, must be transmitted to $P_{i+1}\ldots P_k$. This consumes roughly the same amount of communication time as for the block row partitioning, (4.2) and (4.3). The arithmetic operations during step j take time

$$(N-j)(\gamma + \frac{N}{k}\omega)$$

as each processor forms $N-j$ linear combinations of rows of length $\frac{N}{k}$. The total time for arithmetic operations comes to

$$t_A \approx \frac{1}{2}\frac{N^3}{k}\omega + \frac{1}{2}N^2\gamma.$$

Unlike (4.4), the start-up time is not reduced by k, since the number of elements each processor has to eliminate per column is independent of k.

Recall that on a sequential machine the time for Gaussian elimination is proportional to $\frac{1}{3}N^3\omega$. In the preceeding schemes, use of k processors will not speed up the computation by a factor of k, no matter how fast the communication, because processors are often idle. There are several ways of improving the efficiency of these algorithms. We could keep processors busy by having idle processors continue the elimination on rows above the pivot row instead of remaining inactive; this is the Gauss-Jordan method, to be discussed later. An alternative is scattering of rows or columns across processors as was done already for the solution of triangular systems.

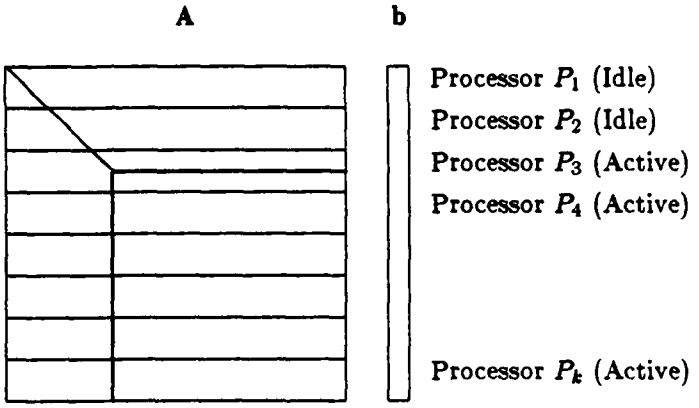


Figure 8: Gaussian Elimination on a Block Row Partitioned Matrix.

4. Gaussian elimination

In this section we describe parallel implementations of Gaussian elimination on a dense $N \times N$ matrix A for solving the linear system

$$Ax = b. \tag{4.1}$$

It is assumed that no pivoting is required. The issue of pivoting will be discussed in the next section.

4.1. Partitioning the Matrix into Blocks of Contiguous Rows

The simplest way to implement Gaussian elimination is to subdivide the matrix A into k blocks of $\frac{N}{k}$ rows each and assign one such block to each processor, cf Section 3.1. Let processor P_i hold rows $(i-1)\frac{N}{k}+1$ to $i\frac{N}{k}$ of A and the corresponding components of the right hand side vector b, see Figure 8.

If at step j, row j is stored in P_i, it must be sent to $P_{i+1} \dots P_k$ in order to perform the eliminations in each of them. In section 2 it was shown that if only the local links are used, then the transfer of a row, j, of length $N-j$ to $\tilde{i} \equiv k-i$ processors approximately requires time

$$\left(\sqrt{(N-j)\tau_L} + \sqrt{\beta_L \tilde{i}}\right)^2,$$

where $\tilde{i}$ depends on j

$$\tilde{i} = k - \left\lceil \frac{j}{N/k} \right\rceil \approx k(1 - \frac{j}{N}).$$

This yields an approximate communication time for step j of

$$(N-j)\left(\sqrt{\tau_L} + \sqrt{\frac{k\beta_L}{N}}\right)^2.$$

After summation from $j = 1$ to $N-1$, this yields

$$t_{T,L} \approx \frac{N^2}{2}\left(\sqrt{\tau_L} + \sqrt{\frac{k\beta_L}{N}}\right)^2,$$

Method	t_A	$t_{T,L}$	$t_{T,B}$
TRB	$\frac{3N^2}{2k}\omega + 2N\gamma$	$2N\tau_L + k^2\beta_L$	$N\tau_B + k\beta_B$
TCB	$(\frac{3N^2}{2k} + \frac{N}{k})\omega + (\frac{k^2}{2} + 2N)\gamma$	$\frac{kN}{2}\tau_L + \frac{k^2}{2}\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TDB	$\frac{3N^2}{2k}\omega + 2N\gamma$	$3N\tau_L + \frac{k^2}{2}\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TRB/DDB	$\frac{N^2}{k}\omega + 2N\gamma$	$2N\tau_L + k^2\beta_L$	$N\tau_B + k\beta_B$
TCB/DDB	$(\frac{N^2}{k} + \frac{N}{k})\omega + (\frac{k^2}{2} + 2N)\gamma$	$\frac{kN}{2}\tau_L + \frac{k^2}{2}\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TDB/DDB	$\frac{N^2}{k}\omega + 2N\gamma$	$3N\tau_L + \frac{k^2}{2}\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TRS	$(\frac{N^2}{2k} + N)\omega + (\frac{N^2}{2k^2} + N)\gamma$	$kN\tau_L + kN\beta_L$	$N\tau_B + N\beta_B$
TCS	$(\frac{N^2}{2k} + \frac{N^2}{2} + 2N)\omega + (\frac{N}{k} + 3N)\gamma$	$\frac{N^2}{2}\tau_L + 2N\beta_L$	$(\frac{kN}{2} + \frac{N^2}{2})\tau_B + (\frac{kN}{2} + N)\beta_B$
TRS/DDB	$\frac{N^2}{2k}\omega + \frac{N^2}{2k^2}\gamma$	$N\tau_L + N\beta_L$	$N\tau_B + N\beta_B$
TCS/DDB	$(\frac{N^2}{2k} + \frac{N^2}{2})\omega + (\frac{N}{k} + N)\gamma$	$\frac{N^2}{2}\tau_L + N\beta_L$	$\frac{N^2}{2}\tau_B + N\beta_B$
TRBG	$\frac{2N^2}{k}\omega + 2N\gamma$	$2N\tau_L + 2k\beta_L$	$kN\tau_B + k^2\beta_B$
TCBG	$\frac{2N^2}{k}\omega + 2N\gamma$	$2N\tau_L + 2k\beta_L$	$kN\tau_B + k^2\beta_B$

Table 1: Computation Times for Several Triangular System Solution Algorithms.

- When the number of processors, k, is much smaller than N (with the exception of the column scattering scheme, TCS) communication times are of lower order than arithmetic times.
- In general, the scattered schemes seem to show better arithmetic than communication performance. This is because fewer processors are idle than in block methods, which in turn results in increased data transfers.
- Broadcasting, it appears, is best done with row-oriented schemes and/or 'greedy' methods. For a small number of processors TRBG and TCBG, and otherwise TRS/DDB are to be preferred.
- Communication on local links is fastest with row-oriented schemes, such as TRB/DDB and TRB, as well as with the diagonal block method TDB. For a large number of processors, $k \approx N$, the row scattering schemes also fare well.
- The merit of an algorithm regarding arithmetic depends very much on the relation of pipe-fill times to elemental operation times and the number of processors. For example, for non-pipelined machines the row scattering scheme, TRS/DDB seems to be most attractive. For large pipe-fill times and a small number of processors the block DDB methods look good.
- The block schemes might be better suited for pipelined machines than the scattering schemes since their coefficients for pipe-fill times are lower.
- DDB matrices do not improve the performance of greedy methods, TRBG and TCBG, since simultaneous matrix-vector multiplications conceal the improvement in triangular system solution.
- Disappointingly, the diagonal block scheme, TDB, did not deliver the expected compromise between block schemes (faster communication) and scattering schemes (faster arithmetic).

To summarize, for fast solution of triangular systems on a multiprocessor ring, row-oriented methods seem to be superior to column-oriented algorithms. In particular, when communicating on local links the block row DDB method, TRB/DDB, and also TDB/DDB, are recommended; if the number of processors is proportional to N, then the row scattering algorithms, TRS and TRS/DDB should also be considered. A row-oriented scheme, TRS/DDB, or greedy scheme, TRBG or TCBG, should be chosen when data transfer is done via broadcasting.

using the local links. However, this expression can be simplified by using the upper bound $2(\frac{N}{k}\tau_L + i\beta_L)$ derived from (2.4), which corresponds to splitting the data into a non-optimal number of packets. Using the broadcast bus, the transfer time is just

$$\beta_B + \frac{N}{k}\tau_B.$$

- In step 2(b), $\frac{N}{k}$ matrix-vector multiplications of length $\frac{N}{k}$, in time

$$\frac{N}{k}(\gamma + \frac{N}{k}\omega).$$

- In step 2(c), sending the result of step 2(b) from one processor to its neighbor can be done using the local links simultaneously for all processors in time

$$\beta_L + \frac{N}{k}\tau_L,$$

but must be sequenced when using the bus,

$$i\beta_B + i\frac{N}{k}\tau_B.$$

- Solving the system (3.27), which requires time

$$\frac{N}{k}(\gamma + \frac{1}{2}\frac{N}{k}\omega).$$

Hence, the communication and arithmetic times for algorithm TDB are

$$t_{T,L} \approx 3N\tau_L + \frac{1}{2}k^2\beta_L$$
$$t_{T,B} \approx \frac{1}{2}kN\tau_B + \frac{1}{2}k^2\tau_B$$
$$t_A \approx \frac{3}{2}\frac{N^2}{k}\omega + 2N\gamma.$$

The arithmetic and local link communication times are comparable to those of the block row method, TRB, while the broadcast transfer time is the same as for the block column algorithm, TCB.

Analogously to the other block schemes, for a DDB matrix only the arithmetic time is reduced, to

$$t_A \approx \frac{N^2}{k}\omega + 2N\gamma.$$

3.8. Summary

The complexity bounds for the various algorithms considered in this section are summarized in Table 1. Under the assumption that high order terms realistically reflect the time behavior of the methods, the following conclusions can be drawn.

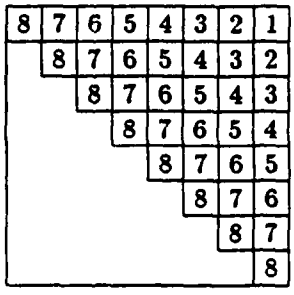


Figure 7: Block Diagonal Scattering of a Triangular System for $k = 8$.

ALGORITHM TDB (Triangular system solution with Diagonal Blocks)

1. Solve in P_k

$$U_{kk}x_k = b_k$$

2. For $i = k-1, k-2, \ldots 1$ do
   (a) Send x_{i+1} from P_k to $P_{k-1}, P_{k-2} \ldots P_{k-i}$
   (b) For $j = k-i, k-i+1, \ldots k-1$ do in P_j $(y_{jk} \equiv b_j)$

$$y_{j+1,i} := y_{j,i+1} - U_{j,i+1}x_{i+1} \tag{3.26}$$

   (c) For $j = 1, 2, \ldots i$ do in P_j
   Send $y_{j+1,i}$ to P_{j+1}
   (d) Solve

$$U_{ii}x_i = y_{k,i} \tag{3.27}$$

At each iteration in the above algorithm the following has to be done

- In step 2(a), transfering the vector x_{i+1} from processor P_k, where it has just been computed, to i other processors, in time

$$(\sqrt{\frac{N}{k}\tau_L} + \sqrt{i\beta_L})^2$$

Adding (3.22) and (3.21) results in an arithmetic time of

$$t_A \approx (\frac{1}{2}\frac{N^2}{k} + N)\omega + (\frac{1}{2}\frac{N^2}{k^2} + N)\gamma. \tag{3.23}$$

In a comparison with methods TRB, TCB and TCBG on the one hand, the coefficient for the elemental operation time ω in TRS is the smallest; on the other hand, the coefficient for the pipe-fill time γ is increased by $\frac{1}{2}(\frac{N}{k})^2$, which makes the contribution of γ in TRS *always* larger than in the other schemes.

3.6. Scattering the Rows of a Diagonal-Diagonal-Block Matrix

Scattering the rows of a DDB matrix decreases both the arithmetic and the communication times. For each of the $\frac{N}{k}$ steps, the time (3.21) can now be subtracted from the total arithmetic time, yielding

$$t_A \approx \frac{1}{2}\frac{N^2}{k}\omega + \frac{1}{2}(\frac{N}{k})^2\gamma. \tag{3.24}$$

Since no triangular system has to be solved, each processor contains exactly *one* element of the vector x_{i+1} in (3.18). Yet, prior to performing the operations in (3.17), the entire vector x_{i+1} must be made available in all processors. Thus, a direct sum of k 'vectors' of length one, as described in section 2 ought to be performed. From (2.6) the transfer time for operations (3.17) is therefore $k(\tau_L + \beta_L)$ per step on the local links. For all $\frac{N}{k}$ steps this makes

$$t_{T,L} \approx N\tau_L + N\beta_L. \tag{3.25}$$

For the broadcast bus, the time is obviously similar,

$$t_{T,B} \approx N\tau_B + N\beta_B.$$

Observe that only here and in algorithm TRS with the broadcast bus, neither the transfer nor the arithmetic time increases with the number of processors, k. Among the methods discussed so far, the contribution of the elemental operation time ω in the DDB scattering scheme is the smallest while that of the pipe-fill time γ is the largest when $k < \sqrt{N}$.

3.7. Partitioning of the Matrix into Block Diagonals

Scattering of block diagonals instead of rows or columns will be considered in this section. The matrix is partitioned into blocks of size $\frac{N}{k} \times \frac{N}{k}$. As before, U_{ij} represents the block matrix in position (i,j) of U. The matrix is scattered so that processor P_{k-i} contains block superdiagonal i of $U, i = 0,..k-1$; in particular, the main block diagonal (superdiagonal 0) is contained in P_k.

Formally, P_{k-i} contains block matrices $U_{j,j+i}$, for $j = 1,2,..,k-i$. This scheme is described in Figure 7, where the matrix entries denote the processor in which the corresponding block matrix is contained. Initially, corresponding elements of the right hand side b and the last column of U reside in the same processors. The triangular system can then be solved with algorithm TDB (Triangular system solution with Diagonal Blocks).

ALGORITHM TRS (Triangular System Solution with Scattered Rows)

1. Solve

$$U_{mm}x_m = b_m, \qquad \text{where} \quad m = \frac{N}{k}$$

2. For $i = \frac{N}{k} - 1, \frac{N}{k} - 2, \ldots 1$ do

   (a) For $j = 1, 2, \ldots i$ do

$$b_j := b_j - U_{j,i+1}x_{i+1} \tag{3.17}$$

   (b) Solve

$$U_{ii}x_i = b_i. \tag{3.18}$$

There are two main tasks in the loop of the above algorithm, solving the $k \times k$ triangular systems (3.18) and performing the matrix-vector multiplications (3.17).

Note that during the triangular system solution each row of the matrix U_{ii} system is contained in a different processor. Therefore one can use the results of Section 3.1 with $N = k$, i.e. $\frac{N}{k} = 1$. Observe, that it is most efficient to send each newly found element of the solution vector directly to all other processors. However, since only one scalar at a time is transmitted, the data transfer time comes to $k(\tau_L + \beta_L)$ for the local links; for each triangular system the amount of transfers comes to

$$k^2(\tau_L + \beta_L).$$

Since broadcasting does not depend on the number of processors to be addressed, the solution of (3.18) requires a number of data movements proportional to

$$k(\tau_B + \beta_B).$$

For $\frac{N}{k}$ linear systems the communication cost is thus

$$t_{T,L} \approx Nk(\tau_L + \beta_L) \tag{3.19}$$

and

$$t_{T,B} \approx N(\tau_B + \beta_B). \tag{3.20}$$

Once the system (3.18) is solved, each processor contains all known elements of the solution vector. In contrast to the previous schemes TRB, TCB and TCBG, which partition the matrix into contiguous parts, the coefficient of the start-up times β in the scattering scheme grows with the problem size (for broadcast bus as well as local links).

The arithmetic time for the solution of a $k \times k$ triangular system is $k(\gamma + \omega)$, resulting in a total of approximately

$$N(\gamma + \omega). \tag{3.21}$$

Once the vector x_{i+1} is available in all processors, each processor will subtract from its component of b_j the inner-product of its row of U with x_{i+1} in time $\gamma + k\omega$, $j = 1, 2, \ldots i$. Hence the arithmetic time in step i of algorithm TRS is $i(\gamma + k\omega)$ and the total over all steps in (3.17) comes to

$$\frac{1}{2}\left(\frac{N}{k}\right)^2(\gamma + k\omega). \tag{3.22}$$

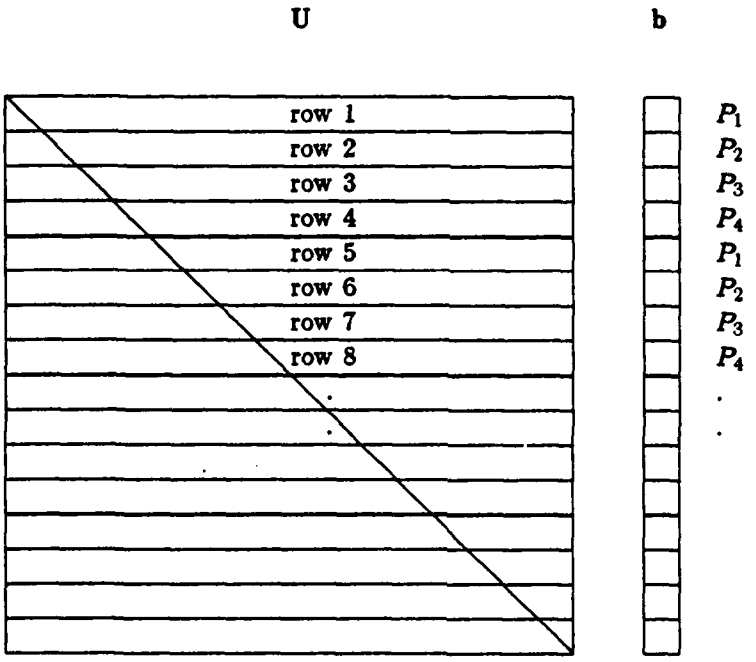


Figure 6: Scattering the Rows of a Triangular System for $k = 4$.

Note, that the computation time is not diminished when TCBG is applied to a DDB matrix, because the simultaneous matrix-vector multiplications conceal the improvement in the triangular system solution.

3.5. Scattering the Rows of the Matrix

The previous algorithm is arithmetically not efficient since many processors are idle during an important part of the process. A remedy is to simply scatter the rows of the matrix U across the processors in a cyclic way so that the work is divided more evenly and processors become idle only during the last k steps (this is termed 'torus wrap' in [5]). Clearly, one can expect the communication time to increase somewhat, while the arithmetic time should decrease.

Let the rows of U be scattered in such a way that P_i contains rows $i, i+k, i+2k, \ldots i+(\frac{N}{k}-1)k$. This time the matrix U is divided up into blocks of size k each (separated by bold lines in Figure 6) instead of $\frac{N}{k}$ as in Section 3.1. Let $b_1, b_2, \ldots, b_{N/k}$ and $x_1, x_2, \ldots, x_{N/k}$ be the blocks of the right hand side b and the solution x, respectively, corresponding to the above partitioning. After all processors have participated in the triangular system solution, they perform matrix-vector multiplications with the newly found part of the solution vector which each of them contains. Algorithm TRB is modified as follows.

Proceeding from bottom to top of the matrix, $i = k, k-1, \ldots 1$, processor P_i solves the $\frac{N}{k} \times \frac{N}{k}$ triangular system associated with the ith diagonal block and then, successively, performs a matrix-vector multiplication with x_i and each matrix block 'higher up' in the column. In the mean time, the neighboring processor to the left can start solving its triangular system, followed by matrix-vector multiplications of all matrix blocks in the corresponding column with the solution vector part.

All steps of Algorithm TCBG consist of a triangular system solution followed by matrix-vector multiplications and additions, after each of which at most $k/2$ processors *simultaneously* transfer a vector of length $\frac{N}{k}$ via the local links to their left neighbor in time

$$2\frac{N}{k}\tau_L + 2\beta_L.$$

Since this communication takes place only between *pairs* of processors, the vector is not divided into packets of smaller size and the number of start-up times is reduced by more than a factor of k compared to TRB and TCB. After $k-1$ steps the time for data communication is about

$$t_{T,L} \approx 2N\tau_L + 2k\beta_L.$$

In case of broadcasting no simultaneous transfer is possible anymore and the upper bound per step increases to

$$2\frac{k}{2}(\frac{N}{k}\tau_B + \beta_B) = N\tau_B + k\beta_B,$$

bringing the total time for data exchange to

$$t_{T,B} \approx kN\tau_B + k^2\beta_B,$$

which, as expected, exceeds by a factor of k the communication time involving local links.

The communication time in TCBG with local links is superior by a factor of k to the one in TCB. If it comes to broadcasting, TRB is the preferred scheme. Hence, the row-oriented scheme would benefit from broadcasting while the column-oriented scheme would be better off with data exchange on the local links.

For the arithmetic operation time, note that in contrast to TCB, matrix-vector multiplications and linear system solutions are overlapped. As the processors are assumed to work in lockstep and a matrix-vector multiplication needs twice the amount of time of a triangular system solution of the same size, the arithmetic operation count is proportional to

$$t_A \approx 2\frac{N^2}{k}\omega + 2N\gamma, \qquad (3.15),$$

which is larger than the one for TRB or TCB. Moreover, at any given time never more than half of the processors are active, cf Figure 5.

3.4. Partitioning a Diagonal-Diagonal-Block Matrix into Contiguous Blocks of Rows or Columns

The solution of the triangular systems in TRB and TCB can be avoided and the arithmetic time reduced by about 50% when the diagonal blocks of U are $\frac{N}{k} \times \frac{N}{k}$ diagonal matrices. Such matrices are referred to as diagonal-diagonal-block (DDB) matrices. The communication times remain the same while the arithmetic time for both TRB and TCB is reduced to

$$t_A \approx \frac{N^2}{k}\omega + N\gamma \qquad (3.16).$$

Note that for the row-oriented schemes, as is the case on a sequential machine, there is no need to actually permute the pivot row with the j^{th} row, i.e. to physically move the two rows to new locations among the processors. All that is needed is to associate with each row an integer, indicating the actual position of this row with respect to the resulting upper triangular system. The result will be an upper triangular system whose rows are scattered in a random way accross the processors, each processor holding N/k rows. However, this will not only complicate the organization of the solution of the resulting triangular system but will also increase the communication time.

To be concrete let us outline an analogue of the TRB algorithm of Section 3.1 for solving an upper triangular system whose rows are randomly scattered as a result of the above pivoting technique. The first step consists in computing the last component ξ_N of the solution x (cost $= \gamma + \omega$), in the processor holding the last row (therefore each processor must check whether it contains the last row but we neglect the cost for these tests). Then the component ξ_N is sent to all processors (cost $= k(\beta_L + \tau_L)$; $(\beta_B + \tau_B)$), which will then perform the analogue of step 2.(b) of Algorithm TRB, i.e. they will subtract from the right hand side their part of the last column of U times the last component ξ_N just received. Since each processor holds at most N/k elements of any column of U, this will take at most $\frac{N}{k}\omega + \gamma$. Next ξ_{N-1} is computed in some processor and the above is repeated. Because of the random scattering of the rows, each component must be sent to all processors at any step. Also, since the number of rows contained in each processor is only known to be at most N/k at any given step, the arithmetic time required for each step is upper bounded by $\frac{N}{k}\omega + \gamma$. If we sum up over N steps, the total time for solving such a randomly scattered triangular system comes to

$$t_{T,L} \approx kN(\tau_L + \beta_L)$$

or

$$t_{T,B} \approx N(\tau_B + \beta_B)$$

for communication and

$$t_A \leq (\frac{N^2}{k} + N)\omega + 2N\gamma,$$

for arithmetic. Observe the increase of the contribution from latencies in the communication time when the local links are used.

The complexity of the algorithm itself is a non-negligible difficulty. However, the column schemes do not present these drawbacks since the rows can be permuted without any data movement. The Gauss-Jordan algorithm will not lead to the above difficulties as the resulting system is diagonal, but it will remain potentially unstable. In view of the fact that the above costs for solving triangular systems are small in comparison with those of Gaussian elimination, it appears that on the whole the column scheme is by and large more attractive when pivoting is necessary.

6. Discussion

The performances of the Gaussian elimination algorithms considered in Section 4 are summarized in Table 2. The following observations can be made by examining the results of Sections 3, 4, and 5.

- The communication times are low order terms as compared with arithmetic times when $k << N$.
- Both the arithmetic times and the communication times of those of the triangular system solution algorithms are low order terms in comparison with those of Gaussian elimination.

Method	t_A	$t_{T,L}$	$t_{T,B}$
GE/BR	$\frac{N^3}{2k}\omega + N^2\gamma$	$\frac{N^2}{2}\tau_L(1+\alpha)^2$	$\frac{N^2}{2}\tau_B + N\beta_B$
GE/BC	$\frac{N^3}{2k}\omega + \frac{1}{2}N^2\gamma$	$\frac{N^2}{2}\tau_L(1+\alpha)^2$	$\frac{N^2}{2}\tau_B + N\beta_B$
GE/RS	$\frac{N^3}{3k}\omega + \frac{N^2}{2k}\gamma$	$N^2\tau_L(1+\frac{8}{3}\alpha+2\alpha^2)$	$\frac{N^2}{2}\tau_B + N\beta_B$
GE/CS	$\frac{N^3}{3k}\omega + \frac{N^2}{2}\gamma$	$N^2\tau_L(1+\frac{8}{3}\alpha+2\alpha^2)$	$\frac{N^2}{2}\tau_B + N\beta_B$
GE/DDB	$\frac{N^3}{3k}\omega + \frac{N^2}{2k}\gamma$	$N^2\tau_L(1+\frac{8}{3}\alpha+2\alpha^2)$	$\frac{N^2}{2}\tau_B + N\beta_B$
GE/DS	$\frac{N^3}{2k}\omega + \frac{N^2}{2}\gamma$	$N^2\tau_L + Nk\beta_L$	$\frac{N^2}{2}\tau_B + kN\beta_B$
GJ	$\frac{N^3}{2k}\omega + N^2\gamma$	$N^2\tau_L(1+\frac{8}{3}\alpha+2\alpha^2)$	$\frac{N^2}{2}\tau_B + N\beta_B$

Notes:

1) $\alpha = \sqrt{\frac{k\beta}{N\tau}}$

2) We have the following upper bounds:

$\frac{N^2}{2}\tau_L(1+\alpha)^2 \leq N^2\tau_L + kN\beta_L.$

$N^2\tau_L(1+\frac{8}{3}\alpha+2\alpha^2) \leq N^2\tau_L + 2kN\beta_L.$

Table 2: Timings for several Gaussian elimination algorithms

- The scattered schemes have better arithmetic performances but worse communication performances.
- The communication times are better with the bus than with local links, which is to be expected since pivot rows must be broadcast to several processors at any given step.
- The diagonal scattering of data results in poor overall performance.
- The overhead of pivoting is small as compared with the cost of Gaussian elimination. It is however of the same order of magnitude as that of triangular system solution.
- Pivoting is less expensive for the row-oriented schemes.

References

[1] D. Gannon, J. van Rosendale, *On the Impact of Communication Complexity in the Design of Parallel Algorithms*, Technical Report 84-41, ICASE, 1984.

[2] W.M. Gentleman, *Some Complexity Results for Matrix Computation on Parallel Processors*, JACM., 25 (1978), pp. 112–115.

[3] D.E. Heller, *A Survey of Parallel Algorithms in Numerical Linear Algebra*, SIAM Review, 20 (1978), pp. 740–777.

[4] D. Lawrie, A.H. Sameh, *The Computation and Communication Complexity of a Parallel Banded Linear System Solver*, ACM-TOMS, 10/2 (1984), pp. 185–195.

[5] D.P. O'Leary, G.W. Stewart, *Data-Flow Algorithms for Parallel Matrix Computations*, Technical Report 1366, Dept. of Computer Science, University of Maryland, 1984.

[6] D.A. Reed, M.L. Patrick, A Model of Asynchronous Iterative Algorithms for Solving Large, Sparse Linear Systems, *Proceedings of the 1984 International Conference on Parallel Processing*, Bellaire, Michigan, 1984, pp. 402–409.

[7] ———, *Parallel, Iterative Solution of Sparse Linear Systems : Models and Architectures*, Technical Report 84-35, ICASE, 1984.

[8] A.H. Sameh, *On Some Parallel Algorithms on a Ring of Processors*, Technical Report , University of Illinois at Urbana-Champaign, 1984.

[9] ———, Numerical Parallel Algorithms - A Survey, D. Lawrie, A. Sameh ed., *High Speed Computer and Algorithm Organization*, Academic Press, New York, 1977, pp. 207–228.

[10] A.H. Sameh, D.J. Kuck, Parallel Direct Linear System Solvers - A Survey, *Parallel Computers - Parallel Mathematics*, International Association for Mathematics and Computers in Simulation, 1977, pp. 25-30.

[11] M.H. Schultz, *XTRAP: Experimental Tandem Rings of Array Processors*, Technical Report , Dept. of Computer Science, Yale University, 1985. In Preparation.

END

FILMED

4-85

DTIC