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 analyzed 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
August 1985

Revised version. 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. For a particular task it is now important in which processor its input data are situated (i.e., how long it takes to move them to the requesting processor) and when they are available. This in turn implies a great richness in the class of algorithms, in terms of the assignments of tasks to processors and the underlying topology of the interprocessor communication network. The algorithms to be presented differ in the way the matrix A and the right-hand side vector b are distributed among the processors.

1.1. Overview

The approach taken here for the development and analysis of algorithms acknowledges that times for data 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 characterized 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 Gaussian elimination without (Section 4) and with partial pivoting (Section 5). A short summary of the main results can be found in Section 6. To begin with, 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 the derivation of arithmetic and communication times will contain merely high order terms (in the size N of the matrix and the number k of processors). Furthermore, only a few representative methods will be described in detail to illustrate their analysis, while others, 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, 8, 10]. Probably the earliest paper to realize 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 analyzed 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 [9]. 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.

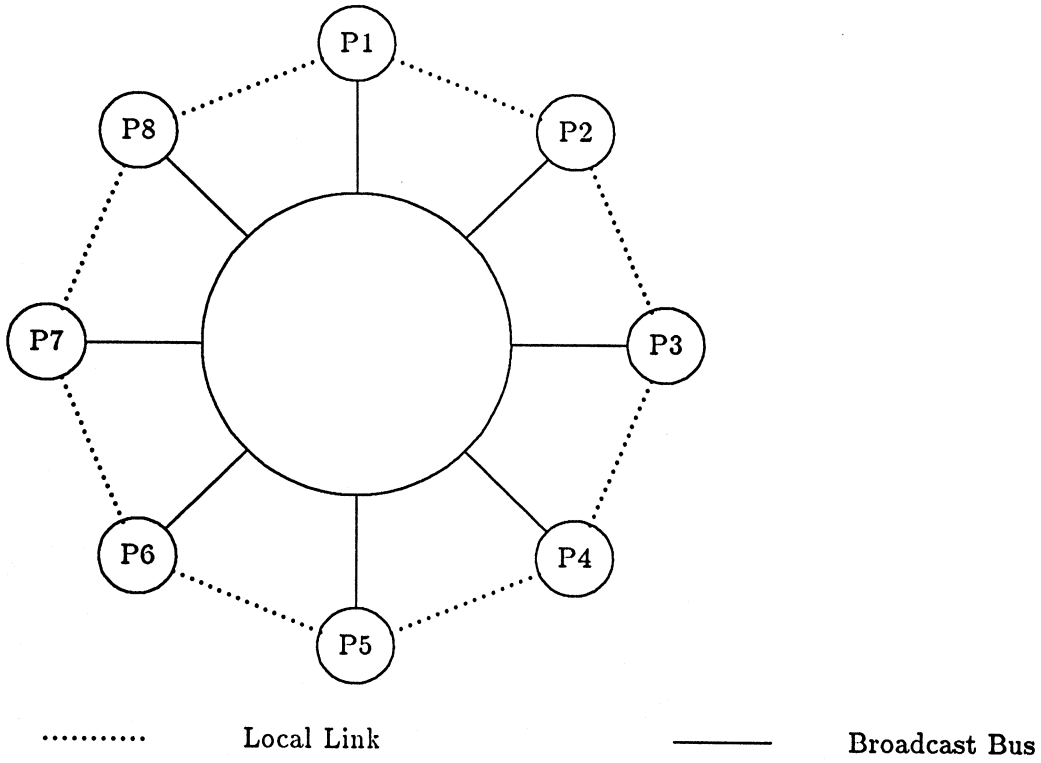

Figure 1: Ring of $k = 8$ Processors.

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 analyze, for the solution 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

- 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 connected to all processors
- local interconnections linking each processor to its two nearest neighbors (the ring).

The following assumptions will be made throughout the paper. Using the local links, each processor is capable of writing to one neighbor while reading from the other. In order to compare the merits of nearest neighbor communication on one hand and broadcasting on the other, an algorithm will make exclusive use of only one of them, but never use a combination of both local links and the bus. For purposes of estimating the computation time processors are viewed as working in lock-step, that is, they are able to synchronize parallel tasks which ideally would require the same amount of time to complete (this assumption is well justified as the parallel tasks of our algorithms are almost identical). Although the architecture proper is not necessarily SIMD, the algorithms under consideration are of SIMD type and one can consequently regard the stream of instructions as being synchronized. Subject to the above assumptions, our model will yield a good estimate for the total elapsed time on a general loosely coupled distributed system.

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 the sequel, $t_{T,B}$ denotes the data transfer times for the bus, $t_{T,L}$ refers to the one for the local links, and t_A stands for the arithmetic time. For any algorithm, the sum of its transfer and arithmetic time, $t_{T,*} + t_A$, is simply called its *computation time*. Whenever convenient, the size N of the matrix is a multiple of the number of processors k.

With regard to algorithms for the solution of dense linear systems, it will be shown that the two modes of data transfer, broadcasting and pipelined data transfer via nearest neighbor connections, do not result in obviously different computation times, provided the number of processors is not too large. Even though the expressions for the computation times differ in the number of start-ups, they contain data transfer times only as coefficients of low order terms in N, the order of the matrix. Hence data communication has asymptotically no influence on the high order terms which are relevant for the overall time estimate.

For this reason, we will consider algorithms where communication is not overlapped with computation. In fact, most outer loops of the proposed methods consist of a parallel communication task followed (or preceded) by a parallel computation task, and as a result no processor performs computations while data transfers are taking place. Moreover, most of our methods require broadcast-type data transfers so that after a short start-up time all communication links are active during the data transfer phase of the loop. In case the processors are not provided with I/O co-processors, the arithmetic units are idle while I/O (broadcasting or nearest neighbor communication) takes place.

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 data must be sent to fewer than $k-1$ processors, this scheme becomes more complicated. To achieve optimality, vectors might need to be partitioned into packets of two different sizes, one for the left path and one for the right path, in case the number of processors to be reached to the right and left of P_1 are different. 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 $1 \le \nu \le N$, so that formula (2.2) is valid only when

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

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 $\sqrt{N/k}$. We will sometimes 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 at a distance not exceeding $\tilde{k} < k$ 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 case a vector v is 'uniformly' distributed over the k processors so that the sub-vector v_i of length 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 sub-vectors 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.

3. Solution of Triangular Systems

In sequential machines, a general dense linear system

$$Ax = b,$$

where A is a real N x N matrix, is efficiently solved by first reducing it to triangular form via 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.

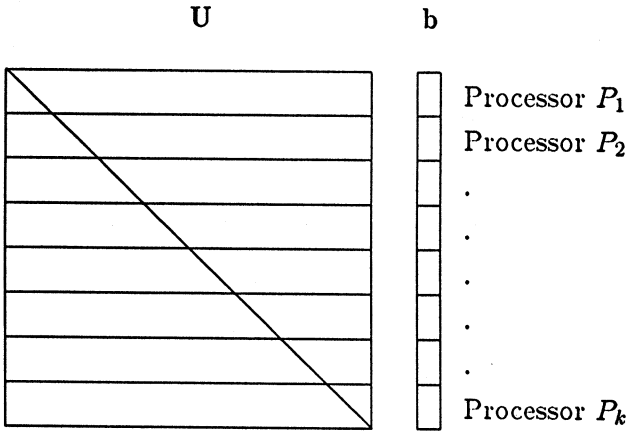

Figure 2: Block-Row Partitioning of a Triangular System.

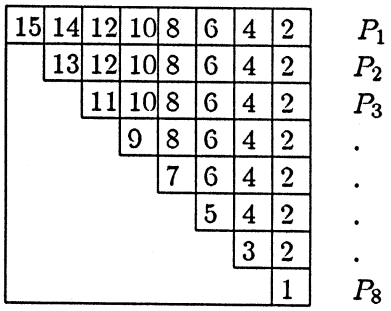

Figure 3: Sketch of Algorithm TRB for $k = 8$.

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 through 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, processor P_i solves the $N/k \times N/k$ triangular system whose coefficient matrix is the ith diagonal block U_{ii}, $i = k, k-1, \ldots 1$. 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.

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 (2.4) and (2.5), requires time

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

using the local links and

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

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

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

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 step i in (3.2) each active processor computes a component of the new vector b_j in N/k pipelined operations which takes time

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

summing this over N/k results in time

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

The cost of solving the triangular system in (3.3),

$$Rx = f,$$

on a pipelined machine requires time

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

because x_i, $1 \leq i < N/k$, 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 divisions are significantly more expensive than additions or multiplications. Summing (3.9) and (3.10) over $k-1$ steps gives as high order terms for the arithmetic time

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

3.2. Partitioning the Matrix into Blocks of Contiguous Columns

In a block column-oriented partitioning scheme each of the k processors contains 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 $N/k \times N/k$ 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 sub-system of order $N/k \times N/k$ 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.

ALGORITHM TCB (Triangular system solution with Block Columns)

1. Solve in P_k

$$U_{kk}x_k = b_k$$

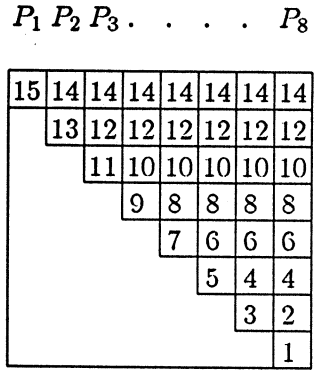

Figure 4: Sketch of Algorithm TCB for $k = 8$.

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
   { Comment : Every processor sends to its left neighbor the block $y_{i,*}$ it just received from its right neighbor. }
   In Processor P_l, $l = i+1, \ldots, j$ send $y_{i,l+k-j}$ to P_{l-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 $y_{i,l+k-j}$ are. The communication time with the local links for 2(b) in step i comes to

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

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

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

The arithmetic time can easily be determined by observing that in each step the matrix-vector multiplications which are all done in parallel are *followed* by $k-i+1$ summations of vectors of length N/k and a subsequent triangular system solution in (3.13). Hence, from (3.9) and (3.10) the time per step is

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

and the total for k steps is given by

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

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.

In contrast with TRB, the vector additions in 2(c) cannot be performed in parallel and hence their computation time is increased by a factor of k. It would seem that this time could be improved by performing summations in a 'tree-like' 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 $k/2$ 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 a tree-like fashion and then to send the partial sums and the remaining $\log(k-i)-f_i$ vectors to *one* processor for the computation of the final sum.

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

In Algorithm TCB *all* processors are waiting for the solution of the triangular system with coefficient matrix U_{ii} 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 the necessary data (corresponding blocks of the solution vector) are available. This leads to the scheme illustrated in Figure 5 where the numbers within each block indicate the sequence of operations. Again, processor P_i comprises columns $(i-1)\frac{N}{k}$ to $i\frac{N}{k}$ of U.

Note that sequencing the tasks based on the availability of the operands does not mean that this algorithm is a data-flow algorithm. It could clearly be implemented in a data-flow fashion, but our complexity analysis would not be valid without the original assumption of synchronization of the outer loops. In Figure 5 this means, for example, that none of the tasks whose number is 7, will be carried out before all the tasks numbered 6 are completed.

The first step of the method consists of solving the triangular system $U_{kk}x_k = b_k$. Once x_k is computed, processor P_k performs the multiplication $U_{k-1,k}x_k$ and subtracts the result from b_{k-1}. It passes on this transformed part of the right-hand side to processor P_{k-1} which uses it to obtain x_{k-1} by solving the triangular system involving $U_{k-1,k-1}$. Meanwhile, processor P_k prepares for the coming steps and computes $U_{k-2,k}x_k$ which it then subtracts from b_{k-2}. In general, processor P_i solves the triangular system with coefficient matrix U_{ii} while processors $P_{i+1}\ldots,P_k$ perform a matrix-vector multiplication involving previously computed parts of the solution vector. The product is subtracted from the corresponding part of the right-hand side, and the result is sent to the processor on the left.

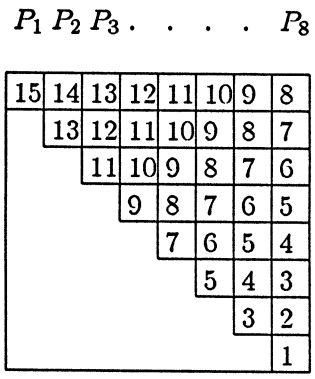

Figure 5: Sketch of Algorithm TCBG for $k = 8$.

There are $2k - 1$ steps in Algorithm TCBG; when i is odd, step i, $i = 1, 2, \ldots, 2k - 1$, consists of a matrix-vector multiplication overlapped with a triangular system solution, which in turn is followed by the simultaneous transfer (via the local links) of a vector of length N/k from at most $k/2$ processors to their left neighbors (when i is even, no triangular system solution occurs). 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 $2k - 1$ steps the time for data communication is about

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

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

$$\frac{k}{2}(\frac{N}{k}\tau_B + \beta_B) = \frac{1}{2}N\tau_B + \frac{1}{2}k\beta_B,$$

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 given by (3.14). As for 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 lock-step 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

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

which is larger than the one for TRB or TCB. In fact, as is seen in Figure 5, no more than half the processors are ever active simultaneously.

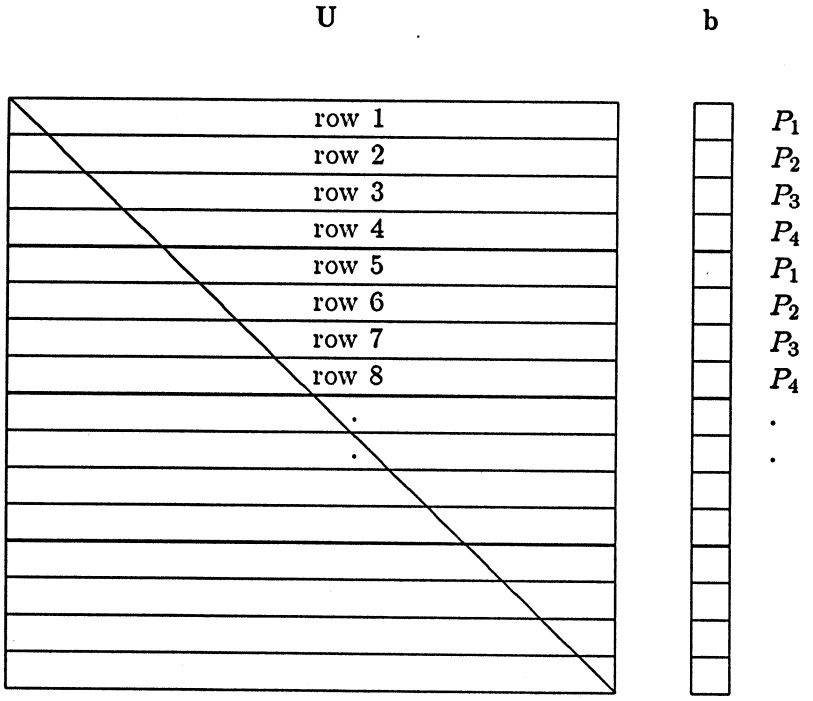

Figure 6: Scattering the Rows of a Triangular System for $k = 4$.

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 $N/k \times N/k$ diagonal matrices. Such matrices are referred to as 'diagonal-diagonal-block' (DDB) matrices. The communication times remain the same while the arithmetic time for TRB is reduced to

$$t_A \approx \frac{N^2}{k}\omega + N\gamma \tag{3.16}$$

and for TCB to

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

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 not efficient with regard to arithmetic 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 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.

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).

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., $N/k = 1$. 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 transfers time is

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

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

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

For N/k linear systems the communication cost is thus

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

and

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

After the solution of system (3.18) 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 $2k(\gamma + \omega)$, resulting in a total of approximately

$$2N(\gamma + \omega). \tag{3.21}$$

The factor of two in (3.21) accounts for the fact that we assume no overlapping of a multiplication and a subsequent division in scalar arithmetic.

Once the vector x_{i+1} is determined in all processors, each processor forms the inner-product of its row of U with x_{i+1} and subtracts the results from its component of b_j 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

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

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

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

On one hand the coefficient for the elemental operation time ω in TRS is the smallest in comparison with methods TRB, TCB and TCBG. On the other hand, the coefficient for the pipe-fill time γ is increased by N^2/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 N/k steps, the time (3.21) can now be subtracted from the total arithmetic time, yielding

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

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 must 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 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,

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

This algorithm and Algorithm TRS with the broadcast bus are the only ones not to show an increase in either communication or arithmetic times as k increases. 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 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 $N/k \times N/k$. As before, U_{ij} represents the block-matrix in

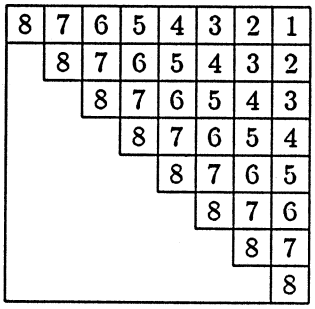

Figure 7: Block Diagonal Scattering of a Triangular System for $k = 8$.

position (i,j) of U. The matrix is scattered so that processor P_{k-i} contains block-superdiagonal i of U, $i = 0,\ldots,k-1$; in particular, the main block-diagonal (superdiagonal 0) is contained in P_k.

Formally, P_{k-i} contains block-matrices $U_{j,j+i}$, $j = 1,2,\ldots,k-i$. 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 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), transferring the vector x_{i+1} from processor P_k, where it has just been computed, to i other processors, in time

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

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), N/k matrix-vector multiplications of length 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 via the local links, in time

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

This is done simultaneously for all processors but must be sequenced when using the bus,

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

- Solving the system (3.27), in time

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

Hence, the communication and arithmetic times for algorithm TDB are

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

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

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

3.8. Summary

The complexity bounds for the various algorithms considered in this section are summarized in Table 1. One might be tempted to believe that a simple look at the table would reveal an optimal algorithm. Although in the purely sequential case it is a perfectly reasonable strategy to seek the fastest method for a given problem, the situation in the parallel context is more subtle. Simply contemplate the complexity of our table: beside the problem size N, the time estimates for algorithms on a multiprocessor ring involve five additional parameters, k, ω, γ, τ, and β. Changing any of these parameters can improve or worsen the speed of a method; in particular, the best method for a specific choice of parameters might be most unsuitable for a different parameter set. This increase in complexity is due to the 'higher dimensionality' of the hardware : apart from the arithmetic speed ω, one has to take into account not only the pipe-fill time γ and the number of processors k relative to the problem size N but also the communication speed governed by β and τ.

It is therefore suggested to interpret the table by considering the coefficients of the various parameters *separately*. When dealing with a small number of processors connected via a broadcast bus, for instance, Algorithm TRB has a communication start-up on the order of $k\beta_B$ while the one for Algorithm TRS is proportional to $N\beta_B$. If the start-up time β_B dominates all other hardware parameters by a large margin, Algorithm TRB is certainly the method of choice. We conclude this section by listing a few observations from the table.

- When the number of processors, k, is much smaller than N communication times are of lower order than arithmetic times (with the exception of the column scattering scheme, TCS).
- 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 perform 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 appealing. For large pipe-fill times and a small number of processors the block DDB methods look attractive.
- 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.

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 Section 5.

4.1. Partitioning the Matrix into Blocks of Contiguous Rows

The simplest way to implement Gaussian elimination is to divide the matrix A into k blocks of N/k rows each and assign one block to each processor as in 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 the jth step row j is stored in P_i then, in order to effect the appropriate eliminations, it must be sent to $P_{i+1} \ldots P_k$. Section 2 showed that the use of local links for the transfer of a row of length $N-j$ to $\tilde{i} \equiv k-i$ processors requires time proportional to

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

where $\tilde{i}$ depends on j via

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

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 $N-1$, one has

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

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 time

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

and therefore the total communication time on the bus is given by

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

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	$(\frac{3N^2}{2k} + \frac{kN}{2})\omega + (\frac{k^2}{2} + 2N)\gamma$	$\frac{kN}{2}\tau_L + \frac{k^2}{2}\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TDB	$\frac{3N^2}{2k}\omega + 2N\gamma$	$3N\tau_L + k^2\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TRB/DDB	$\frac{N^2}{k}\omega + N\gamma$	$2N\tau_L + k^2\beta_L$	$N\tau_B + k\beta_B$
TCB/DDB	$(\frac{N^2}{k} + \frac{kN}{2})\omega + (\frac{k^2}{2} + N)\gamma$	$\frac{kN}{2}\tau_L + \frac{k^2}{2}\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TDB/DDB	$\frac{N^2}{k}\omega + N\gamma$	$3N\tau_L + k^2\beta_L$	$\frac{kN}{2}\tau_B + \frac{k^2}{2}\beta_B$
TRS	$(\frac{N^2}{2k} + 2N)\omega + (\frac{N^2}{2k^2} + 2N)\gamma$	$kN\tau_L + kN\beta_L$	$N\tau_B + N\beta_B$
TCS	$(\frac{N^2}{k} + kN)\omega + (\frac{N^2}{k^2} + 2N)\gamma$	$kN\tau_L + N\beta_L$	$kN\tau_B + N\beta_B$
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	$(\frac{N^2}{k} + kN)\omega + (\frac{N^2}{k^2} + N)\gamma$	$kN\tau_L + N\beta_L$	$kN\tau_B + N\beta_B$
TRBG	$\frac{2N^2}{k}\omega + 2N\gamma$	$N\tau_L + k\beta_L$	$N\tau_B + k\beta_B$
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.

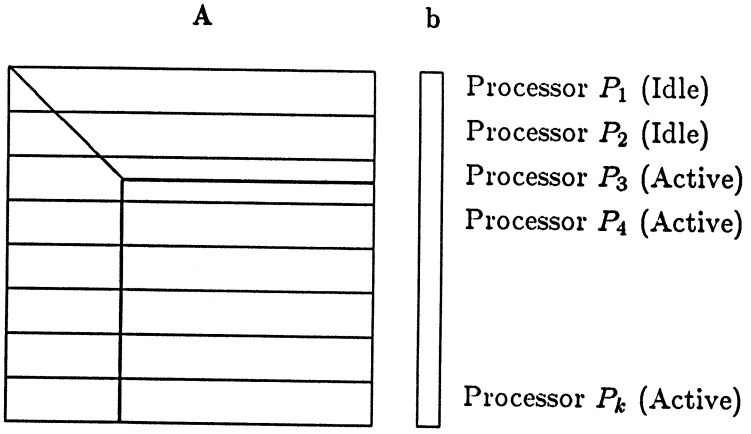

Figure 8: Gaussian Elimination on a Block-Row Partitioned Matrix.

To determine the arithmetic time, we note that at step j a processor performs at most N/k eliminations which takes time

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

and summing over $N-1$ steps,

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

The coefficient start-up time, γ, in the above formula is divided by k because all active processors simultaneously eliminate (at most) N/k elements per column by computing linear combinations of rows of length $N-j$.

4.2. Partitioning 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 and is located 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, given by (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 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 in (4.4), the start-up time is not reduced by k, since the number of elements per column each processor has to eliminate 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 preceding schemes, use of k processors does *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. One could keep processors active by having them

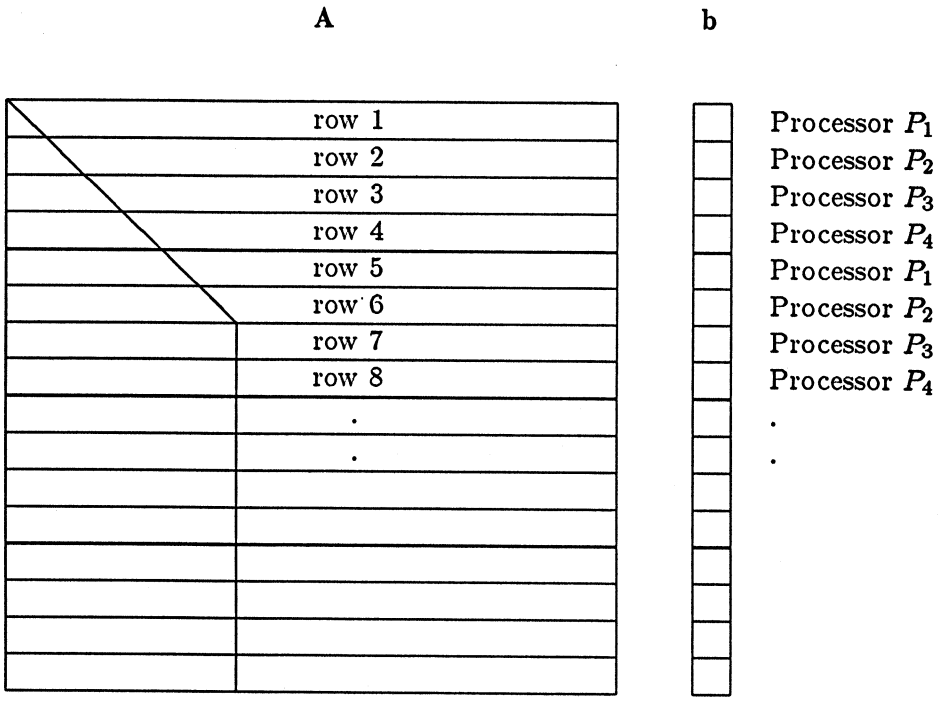

Figure 9: Gaussian Elimination with Scattered Rows for $k = 4$.

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 rows or columns across processors as was done already for the solution of triangular systems.

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.'

An example of row scattering 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), the total communication time comes to approximately

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

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 performs about $\lceil (N-j)/k \rceil$ linear combinations at the cost of $\gamma + (N-j)\omega$ each. Therefore, step j consumes time

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

Using the approximation

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

and summing over j, this yields

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

Observe that, as before, the start-up time is reduced by a factor of k, and 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' version in Section 4.1. For large α, it is roughly twice as big, while for small α the two times are comparable. Furthermore, scattering results in an arithmetic time consistent with the sequential case, that is, for $k = 1$ the arithmetic time reduces to that of the sequential evaluation.

4.4. Reduction to a DDB Scattered Matrix

As mentioned in Sections 3.4 and 3.6, there are advantages to having diagonal matrices in block-positions (i,i) of the upper triangular matrix U. Such DDB matrices can be obtained in the same, parallel, time as regular upper triangular systems, by simply utilizing the idle processors. The result is a triangular system whose solution requires less communication and start-up times, see Sections 3.4 and 3.6.

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 matrix A 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 elements are scattered so that block $A_{i,j}$ belongs to the processor numbered $1 + [(i-j) \ mod \ k]$, $1 \leq i,j \leq k$. The above scheme is illustrated in Figure 10, where a matrix entry denotes the processor to which the corresponding block-matrix is assigned. The right-hand side b constitutes an additional column of A and is distributed accordingly among the processors. Since any two contiguous blocks of A belong to neighboring processors, scattering schemes permit easy overlapping of data transfers across local links.

At each step j of Gaussian elimination, all processors holding a piece of the pivot row (of size $m = N/k$) simultaneously communicate their piece to the processors beneath them. 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 holding the

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.

corresponding piece of the last block-row, which requires exactly $\lfloor (N-j)/m \rfloor$ transfers. Hence, the time for transmitting the jth 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, i.e., the elements of jth column, need to be transmitted to the right. Clearly, this involves the same amount of time as above. The total transfer time thus comes to

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

As data is sent in two directions, from west to east and from north to south, the preceding time is twice as large that of the previous schemes.

In case of broadcasting, each piece of length m can 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 $N-j$ eliminations are performed, each of which consists of taking linear combinations of vectors of length m in different processors. The processor to finish last is P_1 which holds the diagonal blocks and 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

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

It is possible to extend this scheme by scattering diagonals with block size $m \times m$, where $1 \le m \le N/k$, that is, by assigning all the blocks on the same diagonal cyclically to processors $P_1, P_2, \ldots, P_k, P_1, P_2, \ldots, P_k, \ldots$. The parameter m should then be chosen so as to minimize the total computation time. However, we will refrain from discussing this case as it leads to complicated formulas and does not result in a significantly better algorithm.

4.6. Gauss-Jordan Elimination

The Gauss-Jordan algorithm is one of the simplest approaches toward improving the arithmetic efficiency of Gaussian elimination on multiprocessors. Let the matrix A be partitioned into k blocks

of m contiguous rows as shown in Figure 8. As already observed in the discussion of Gaussian elimination in Section 4.1, the idle processors can be employed 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, observe that at each step the pivot row must be sent to all processors. This results in a time identical to that of row scattered Gaussian elimination given by (4.5). Similarly, the arithmetic time is identical to that of Gaussian elimination with block-row partitioning given by (4.4). The obvious advantage of this method over the one 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 because all processors are active during the whole elimination process.

5. Partial Pivoting

So far, the issue of pivoting has been put aside in order to simplify the algorithm description. In fact, it will now be demonstrated that partial pivoting may be incorporated at little extra cost.

First consider the block-row method described in Section 4.1. At the jth step, the element of largest absolute value has to be determined among all elements a_{ij} with $i = j, \ldots N$. This is achieved in two stages. At first, the maximum element in each processor, the 'local maximum', is found. Then all local maxima are compared to obtain the global maximum. The first step costs time $(N/k)\omega'$, where ω' is the time a processor requires to perform one comparison. Taking advantage of nearest neighbor connections in the second step by employing a 'round robin'-type comparison among the local maxima requires a communication time of at most $i(\tau_L + \beta_L)$. Here, $i = k - \lceil j/m \rceil$ is the number of processors involved in the jth step of Gaussian elimination and, at the same time, the number of comparisons needed to determine the global maximum, resulting in a comparison time of $i\omega'$. A similar approach using the bus consists of broadcasting each local maximum in turn to the $i-1$ other processors and performing the comparisons for the global maximum in each of the i processors, in parallel. This leads to similar times for communication and identical times for arithmetic. Summing up the above time estimates over the $N-1$ steps of the Gaussian elimination procedure, 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, respectively, depending on which data transfer mode is used.

For the block-column scheme, the entire jth column resides in one processor, obviating any need for data communication. All comparisons are done in one processor while the others remain idle. Consequently, the overhead time comes to

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

The scattered scheme of Section 3.5 is similar to that of the block-row scheme, except that the first stage involves $\lceil j/k \rceil$ comparisons while the second involves k comparisons. As a result, the overhead time for pivoting is

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

When dealing with row-oriented schemes (as also on a sequential machine), there is no need to actually permute the pivot row with the jth row, i.e., to physically exchange the two rows. One merely needs to maintain a pointer indicating the actual position of a row with respect to the resulting upper triangular system; the rows of this system are scattered arbitrarily across the processors, and each processor contains N/k rows. Consequently, the triangular system solution becomes more complicated and the communication time increases.

For instance, consider what happens in Algorithm TRB of Section 3.1 when the rows of the upper triangular system are randomly scattered, due to the above pivoting technique. To begin with, the processor containing the last row of U computes the last component ξ_N of the solution vector x (cost : $\gamma + \omega$). This implies, however, that each processor has to check whether it indeed contains the last row. To keep matters simple, we will disregard the cost of these tests. The component ξ_N is sent to *all* processors (cost : $k(\beta_L + \tau_L)$ or $\beta_B + \tau_B$), which then perform the analogue of step 2(b) in Algorithm TRB, i.e., they essentially modify the right-hand side vector. Since each processor holds at most N/k elements of any column of U, this takes time at most

$$\frac{N}{k}\omega + \gamma. \tag{5.4}$$

Next, ξ_{N-1} is computed in some processor and the above process repeated. Because of the arbitrary scattering of the rows, each component must be sent to all processors in each step. Moreover, since the number of rows per processor in any step is only known not to exceed N/k, the arithmetic time per step is bounded above by (5.4). Summing over N steps, the arithmetic and communication times, respectively, for solving such a randomly scattered triangular system come to

$$\begin{aligned} t_{T,L} &\approx kN(\tau_L + \beta_L) \\ t_{T,B} &\approx N(\tau_B + \beta_B) \\ t_A &\leq (\frac{N^2}{k} + N)\omega + 2N\gamma. \end{aligned}$$

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

The complexity of the above algorithm ought not to be underestimated. These drawbacks are not encountered when choosing a column scheme, as each processor is able to keep track of the appropriate permutations. The Gauss-Jordan algorithm as well does not lead to such difficulties because the resulting system is diagonal, yet may remain potentially unstable. In view of the fact that the costs for solving triangular systems are small in comparison with those of Gaussian elimination, it appears that on the whole a column scheme is more attractive when pivoting is necessary.

6. Discussion

The performances of the Gaussian elimination algorithms 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 compared to arithmetic times when $k << N$.
- Both, arithmetic and communication times, of the triangular system solution algorithms are low order terms compared to those of Gaussian elimination.
- The scattered schemes show better arithmetic but worse communication performances.
- Communication times are lower when using the bus than with local links (which is to be expected since pivot rows must be broadcast to several processors during each step).

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

Notes:

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

2. We have the following upper bounds:

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

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

Table 2: Computation Times for Several Gaussian Elimination Algorithms.

- Diagonal scattering of data results in poor overall performance.
- The overhead for pivoting is small compared to the cost of Gaussian elimination. It is, however, of the same order as that of triangular system solution.
- Pivoting is less expensive for 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, Numerical Parallel Algorithms - A Survey, D. Lawrie, A. Sameh ed., *High Speed Computer and Algorithm Organization,* Academic Press, New York, 1977, pp. 207–228.

[9] ————, *On Some Parallel Algorithms on a Ring of Processors,* Technical Report , University of Illinois at Urbana-Champaign, 1984.

[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, *Multiple Array Processors for Ocean Acoustics Problems,* Technical Report 363, Computer Science Dept., Yale University, 1985.