Abstract. This paper proposes a few lower bounds for communication complexity of the Gaussian Elimination algorithm on multiprocessors. Three types of architectures are considered: a bus architecture, a nearest neighbor ring network and a nearest neighbor grid network. It is shown that for the bus and the ring architectures, the minimum communication time is of the order of $O(N^2)$, independent of the number of processors, while for the grid it is reduced to $O(k^{-1/2}N^2) + O(k^{1/2}N)$, for a lock-step Gaussian elimination algorithm, and to $O(k^{-1/2}N^2) + O(k^{1/2})$, for any pipelined Gaussian elimination algorithm, where k is the total number of processors. The practical implications of these bounds are discussed.

Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors

Youcef Saad

Research Report YALEU/DCS/RR-348
July 1985

Revised version. Original version issued Jan. 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

The purpose of this paper is to analyse the effects of data movement delays in parallel Gaussian elimination algorithms. In the early research publications on parallel algorithms the communication times were often not accounted for [4], [12] and a parallel algorithm was judged satisfactory whenever the so-called speed-up for the arithmetic was sufficiently close to the number of processors used. It soon became clear that such analyses could be overly simplistic and misleading. Gentleman [3] was probably the first to warn against such abuses by establishing that in parallel computation, communication complexity may be more limitative than arithmetic complexity. Most of the recent analyses of parallel algorithms use timing models that take into account communication times in some form, see e.g. [2, 5, 7, 9, 10].

This paper focusses on the complexity of the Gaussian elimination algorithm, with the viewpoint that the time for transmitting one data item is not negligible relative to the time to perform an arithmetic operation. In [5], several simple parallel Gaussian elimination algorithms have been examined with a systematic incorporation of communication costs in the overall complexity analysis. The architecture considered there consists of a ring of a relatively small number of processors, with nearest neighbor connection and a bus interconnection. It was shown that when the number of processors is small relative to the problem size then the communication time is a low order term as compared with the time to perform arithmetic.

In the present paper we establish further theoretical results concerning the communication complexity of Gaussian elimination on multiprocessors, emphasizing the case when the number of processors is large relative to the size of the problem. In particular, we will attempt to clarify the effect of the architecture on the communication complexity, by putting in contrast the effectiveness of three simple configurations, namely a bus, a ring, and a $2-D$ grid array.

One of our main results is that for the bus and the ring topologies, reducing an N x N system into upper triangular form by the Gaussian elimination algorithm requires a communication time of at least $O(N^2)$, *no matter how many processors are used.* This can be considered as a generalization of a result by Gentleman [3]. As a consequence, even though *arithmetic* speed-ups as high as N^2 can be achieved thus leading to a time linear in N for performing the arithmetic operations, the communication time will never be reduced below the above lower bound of $O(N^2)$.

One reason for this limitation is intuitively clear: besides the "arithmetic" parallelism used to speed-up arithmetic, one also needs "communication" parallelism to speed-up communication. Grid arrays are one way of achieving communication parallelism, since several paths can be used simultaneously. However, as the number of processors increases, a second limitation appears: the distance that a data item must travel will increase on the average thus leading to higher latency times. As a result, it will be seen that when the successive steps of Gaussian elimination are not overlapped, i.e. when it is implemented in a lock-step fashion, then for the $2-D$ connected architectures the lower bound for communication time becomes of the order of $O(k^{-1/2}N^2) + O(k^{1/2}N)$. When some overlapping of the successive steps of Gaussian elimination is assumed, i.e. when these steps are pipelined in some way then the above lower bound becomes $O(k^{-1/2}N^2) + O(k^{1/2})$. An important consequence is that a lock-step Gaussian elimination algorithm cannot achieve a linear time with respect to N in a two-dimensional grid array no matter how many processors are used. In fact the best time that can be achieved is a disappointing $O(N^{5/3})$. On the other hand pipelined implementations of Gaussian elimination can achieve a linear time.

In Section 2 we describe our three models and propose two different general techniques for establishing lower bounds for communication times. In Sections 3, 4 and 5 we propose lower bounds for communication times for Gaussian elimination on the bus, the ring and the grid networks respectively. In Section 6, the bounds for the grid are illustrated on a simple example and in Section 7 a tentative conclusion is drawn.

2. Models and basic properties

2.1. Description of the models

We consider a multiprocessor system consisting of k identical processors, each with its own memory sufficiently large to hold its share of the data. We assume that there is no global memory shared by the processors, or if there is such a memory that it is not used for passing data between processors. Unless otherwise stated, the number of processors does not exceed N^2. The linear system to solve is $Ax = f$, where A is a real N x N matrix. Throughout the paper it is assumed that

- each of the k processors holds $\frac{N^2}{k}$ elements of the matrix A (equidistribution of the data).
- no element of A belongs to more than one processor (exclusivity assumption).

Since we are seeking for lower bounds, the additional commnunication time related to moving the right hand side is neglected for simplicity, so we are not concerned with the way in which the right hand side is distributed among the processors.

There are many different ways in which the processors can be interconnected. In this paper we are interested in the following three simple topologies.

1. The first and simplest architecture consists of a broadcast bus linking the k processors as illustrated in Figure 1. In this model to send a vector of length m from one processor to *any number of processors* via the broadcast bus requires a time of

$$m\tau_B, \tag{2.1}$$

where τ_B will be referred to as the elemental transfer time for the bus, and is expressed in seconds per word.

2. The second architecture consists of a nearest neighbor interconnection ring as shown in Figure 2. To send a vector of length m from one processor to one of its neighbors via the local links requires a time of

$$m\tau_R, \tag{2.2}$$

where τ_R is the elemental transfer time of the local links, in seconds per word. Any processor can send (or receive) a data item from one neighbor while sending (or receiving) another data item from the other neighbor.

3. . The third architecture consists of a grid array of k processors arranged on a square grid with $\sqrt{k}$ processors on each side, see Figure 3. We will often use the term $2-D$ array for grid array. The assumptions are identical with those of the above model but one processor is now able to simultaneously communicate with (at most) four neighbors. The elemental transfer time for the grid is denoted by τ_G.

Ipsen, Saad and Schultz [5] utilize a more general formula for estimating transfer times, in which a (constant) start-up time β is added to the above times. Since we are seeking for lower bounds for communication, we feel that taking $\beta = 0$ is sufficiently representative, though we acknowledge that start-up times may very well dominate in machines with large β.

In this paper we will distinguish between two different implementations of Gaussian elimination. In the first implementation which we will refer to as the lock-step implementation we assume that the successive steps in the outer loop of Gaussian elimination *are not overlapped*, i.e., we consider an implementation that obeys the following structure.

For $j = 1, 2, \ldots, N-1$ do :

(a) *Data Movement:* Read the pivot information (i.e. the j^{th} row and the multipliers) needed by any processor to perform its part of the elimination.

(b) *Arithmetic:* Perform the arithmetic operations of the j^{th} step of Gaussian elimination.

Here the outer loop is sequential but inherent parallelism within each of of the tasks (a) and (b) can be exploited. Some form of synchronization is needed in order to ensure that the next outer step starts only when all processors have completed their substaks for the current step. Ipsen, Saad and Schultz present several examples of implementations that obey this structure in [5]. While this is a restrictive form of Gaussian elimination, good to nearly optimal speed-ups can be achieved for small number of processors and therefore it should not be discarded.

A second, more general, form of Gaussian elimination which we will refer to as pipelined Gaussian elimination allows for the overlapping of successive steps of the Gaussian process. This pipelining can be of fine granularity, e.g. processors perform only one arithmetic operation at a time and pass a few data items to neighboring processors, or can be blocked, i.e., several operations are done in some processor at a time. An example of pipelined algorithms is the wavefront algorithm known for a long time [12, 8, 9].

An important assumption we will make for all three models of architectures and for both types of algorithms, is that arithmetic cannot be overlapped with communication in any processor. For the lock-step implementation for example, each task (step j) of the algorithm is separated into a subtask (a) of data movements, whereby the processors get the pivot information needed to perform the eliminations, and then a subtask (b) of arithmetic computations, and these two subtasks do not overlap. A similar assumption was made by Gannon and Van-Rosendale [2]. Thus one can naturally speak of a communication time (the total time for communication subtasks) and an arithmetic time (the total time for arithmetic subtasks). More generally, the communication time can be defined as the execution time of the given algorithm under the assumption that the times for performing arithmetic are neglected. The arithmetic time is defined in a similar manner, i.e. as the execution time when transfer times are neglected. These definitions, which are consistent with those defined above for the non-overlapping case, have also been used by Gannon and Van-Rosendale [2].

The reason for assuming the non-overlapping of communication and arithmetic is essentially pedagogical and can be justified as follows. Models where arithmetic and communication can be overlapped, are more complicated to analyze. Yet observe that the execution time of an algorithm in an environment where overlapping is assumed is within a factor of *only two* of that of the same algorithm run in an environment where no overlapping is assumed. Indeed, consider any given algorithm and let t_A be the total time to perform arithmetic and t_T be the total time required for data transfers. In the overlapping case the total execution time t_E will be *at least* $\text{Max}\{t_A, t_T\}$ which represents the time with maximum overlapping. In the non-overlapping case it will be simply $t_A + t_T$ which satisfies

$$\frac{1}{2}(t_A + t_T) \leq max\{t_A, t_T\} \leq t_E \leq t_A + t_T.$$

2.2. Two types of lower bounds for communication times

We start by considering a data transfer operation which is essential in Gaussian elimination and which consists in sending a vector of m elements from one processor to all others or to a few neighboring other processors. Using the broadcast bus this type of transfer operation will consume a time of $m\tau_B$ independent of the number of processors to be reached.

For models 2 and 3, an efficient way of achieving this type of data movement, is to pipeline the data as follows. Assume that the data is to be moved from Processor P_1 to the sequence of i consecutive processors $P_2, P_3, \ldots, P_{i+1}$. Then in step 1 the first element is sent from P_1 to P_2. In step 2, while sending the first element to P_3, P_2 receives the second element from P_1. Generally, at

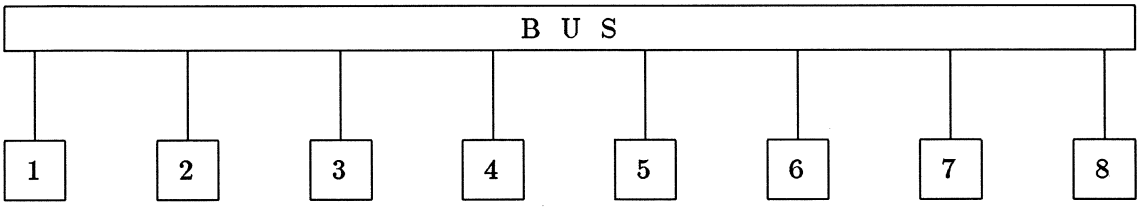

Figure 1: Broadcast bus architecture.

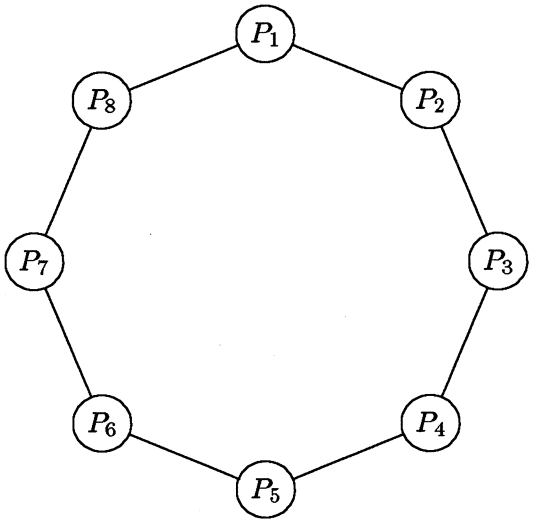

Figure 2: An 8-processor ring.

step j, the first element reaches P_{j+1} from P_j while the second element follows from P_{j-1} to P_j, etc. The first element reaches P_{i+1} at the i^{th} step and $m-1$ more steps are needed for the remaining $m-1$ elements to enter P_{i+1}, resulting in a total of $(m-1)+i$ steps. Therefore, transferring a vector of length m to i consecutive processors takes the time

$$t(m,i) = (m-1+i)\tau, \tag{2.3}$$

where τ stands for either of τ_R (ring) or τ_G (grid). Note that because of the overlapping of data transfers, the above time represents also the time needed for sending a vector of length m from one processor to *all* those that are distant by i local links from it. For the second and third model, the distance between two processors is the minimum number of local links to cross in order to reach one processor from the other.

It is important to distinguish between the two times $(m-1)\tau$ and $i\tau$ in (2.3). The first term $(m-1)\tau$ reflects the actual speed of the local links: the higher the bandwidth of the local links,

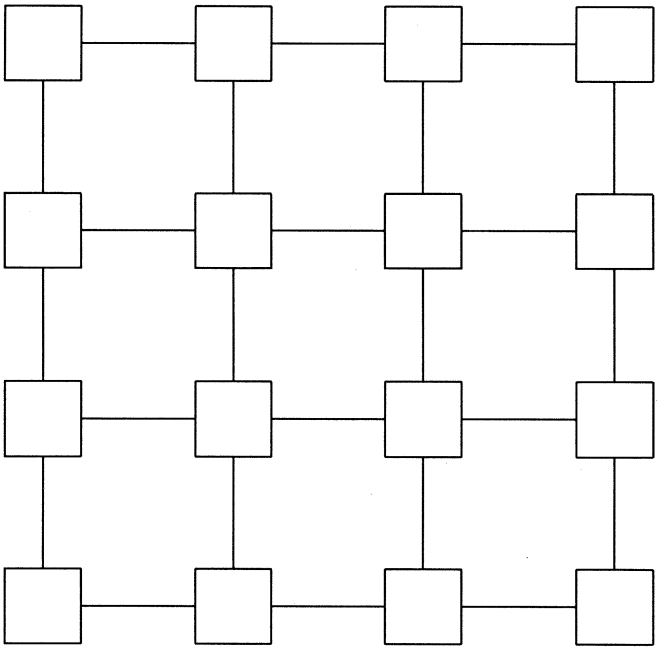

Figure 3: A 4 x 4 multiprocessor grid.

the faster the data transfer, given the same amount of data. The second time is what might be termed latency time, as it corresponds to the time required to fill all i links. The latency time also represents the longest distance that any data item must travel, and is to compare with the start-up times of pipelined arithmetic units.

We now outline how we establish our lower bounds for communication times. For the broadcast bus, simply observe the important fact that sending a vector of length m to *any number of processors* contributes a time of $m\tau_B$ to the total communication time. Thus, in order to derive a lower bound for the communication time for the bus model we only need to count the number of occurances of elemental data transfers from one processor to at least another one. It is very important to note that the lower bounds thus derived *do not depend on which form of Gaussian elimination is used.*

For the nearest neighbor interconnections (Models 2 and 3), it is not as simple because transfers in several edges of the array can take place simultaneously. Also the distance which a data item travels, i.e. the number of edges that it crosses, is now important.

One way to handle these difficulties is to count the total number n of elementary transfers from any one processor to a neighbor. In other words n represents the total over all edges of the number of times that these edges are crossed. Since there are only k edges for the ring and $2(k - \sqrt{k})$ edges for the square grid, a lower bound for communication times will be given by $\frac{n}{k}\tau_R$ for the ring and $[n/(2k - 2\sqrt{k})]\tau_G$ for the grid. We will refer to these lower bounds as the "bandwidth lower bounds" as they take into consideration the *total bandwidth* of the system. It is very important to note that this type of lower bounds *are independent of the form of Gaussian elimination used.* Indeed, whether we use a lock-step or a pipelined Gaussian elimination, we must move the same

amount of data during the whole execution: only the order in which these data movements are performed changes. Therefore, given a certain total bandwidth of the system, the minimum time, as incurred by bandwidth limitation, for performing the data exchange during the algorithm, will be the same.

As an example consider the application of this reasoning to the problem of sending a vector of length m from Processor P_1 to all processors in a k-processor ring. Each of the m elements must cross $k-1$ edges, so the total of elementary transfers is $n = m(k-1)$. From the above argument, it will take a minimum of $\frac{n}{k}\tau_R = m(1-1/k)\tau_R$ to achieve the data transfer. This bandwidth lower bound is nothing but the best possible time in which the n elementary transfers could be realized, if we could keep all k edges saturated during the whole data transfer. However, notice that as was explained earlier this saturation state will not be reached before a start-up time of $(k-1)\tau_R$. Therefore, the lower bounds obtained by this technique does not account for the latency time which is $(k-1)\tau_R$ in this example.

This leads us naturally to a second mechanism for obtaining lower bounds for transfer times, which consists in simply considering the latency time as a lower bound for the total communication time. In the above example this would lead to the lower bound $(k-1)\tau_R$ which may indeed be sharper than the above bandwidth lower bound when k is large with respect to m. This second type of lower bounds will be referred to as the "latency bounds" for obvious reasons. Latency lower bounds are likely to be sharper if the number of processors is large with respect to the data length, while the bandwidth bounds will be sharper if the number of processors is small. As will be seen, a good lower bound should take into account both types of bounds. Gentleman [3] uses a model where $k = N^2$, i.e. each processor holds one element of the matrix, and his lower bounds are latency type lower bounds. Contrary to the bandwidth lower bounds, it is clear that the latency bounds *will depend on which form of Gaussian elimination is used.* This is because by overlapping successive the steps in the outer loop of Gaussian elimination, part of the latency times can be masked by other tasks being executed.

3. Communication complexity for the broadcast bus architecture

In this section, a lower bound for the communication time is derived for the Gaussian elimination algorithm without pivoting implemented on a broadcast bus network. Recall that the fundamental assumptions are that the matrix A is equidistributed among the k processors, and that no data is to be in more that one processor (exclusivity assumption). Note that the matrix A can be assigned in any possible fashion provided each processor holds N^2/k elements.

As was already stated in Section 2, moving a group of m elements to any number of processors will add a time of at least $m\tau_B$ to the total communication time no matter which form of Gaussian elimination is used. We can therefore use an element by element approach to get the desired lower bound for the communication complexity for *any form* of Gaussian elimination: communication time is at least τ_B times the number of elements that must be moved to at least one other processor during the algorithm . The following lemma establishes a lower bound for this number.

Lemma 3.1. *During the Gaussian elimination algorithm the number n_B of matrix elements which must be moved from one processor to at least one other processor satisfies:*

$$n_B \geq \frac{N(N+1)}{2} - \frac{h(h+1)}{2},$$

where

$$h = \lfloor 2N/\sqrt{3k} \rfloor.$$

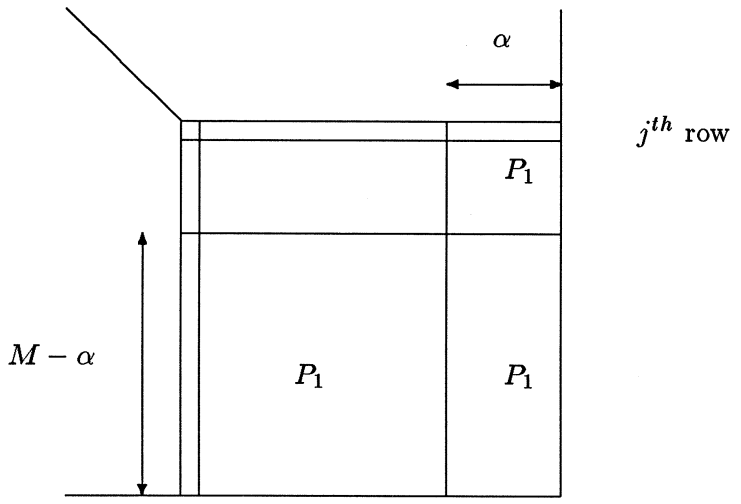

Figure 4: The j^{th} step of Gaussian elimination.

Proof. Consider the j^{th} step of the algorithm as depicted in Figure 4. We assume that $j \leq N - h$ for reasons explained later. Any mention of row (or column) in this proof will refer to the part of the row (or column) of A consisting of its last $(N - j + 1)$ elements.

Any element $a_{j,l}, l = j, j+1, ..N$ must be known to all processors that hold at least one element of the column l below $a_{j,l}$. Similarly, any given element $a_{i,j}, i = j + 1, ..N$ must be known to the processors that hold at least one element of row i, at the right of $a_{i,j}$, in order that these processors be able to compute the pivots.

We would like to show that at least $M \equiv N - j + 1$ elements among those elements of the j^{th} row and column must be moved at step j. Assume that α elements of row j are *not* moved anywhere. If $\alpha = 0$ there is nothing to prove, so assume $\alpha \neq 0$. If an element $a_{j,l}$ of the row is not moved, this means that the whole l^{th} column below that element belongs to the same processor.

We have to show that in this situation at least α elements must be moved from *column j* to the right. Assume that this were not the case, i.e. that at most $\alpha - 1$ of the $M - 1$ elements $a_{i,j}, i = j + 1, ..N$ are moved or equivalently that at least $M - \alpha$ elements on that column are *not moved.* For each of these elements, the corresponding row, once again, belongs entirely to one processor, say P_1. We claim that all of the α "unmoved" rows and the $M - \alpha$ "unmoved" columns must belong to this same processor P_1. This follows easily from the exclusivity assumption: if the whole of row i belongs to one processor, say P_1, while the whole of column l belongs to one processor, say P_2, then $P_1 = P_2$, because $a_{i,l}$ is a commun element to P_1 and P_2.

Now P_1 contains at least

$$(M - \alpha)^2 + M\alpha \geq \frac{3}{4}M^2$$

elements. This would contradict assumption 1 whenever $\frac{3}{4}(N-j+1)^2 > N^2/k$ which is the case because for $j \leq N-h$ we have

$$N-j+1 \geq \lfloor 2N/\sqrt{3k} \rfloor + 1 > 2N/\sqrt{3k}.$$

This explains our initial restriction on j.

We have therefore proved that for each step j such that j satisfies $j \leq N-h$, at least $N-j+1$ elements must be moved. Summing up for $j = 1, 2, ..N-h$ we get the desired lower bound for the number n_B of elements which are moved from one processor to at least one other processor during the algorithm:

$$n_B \geq \sum_{j=1}^{N-h}(N-j+1) = \sum_{i=h+1}^{N} i = \sum_{i=1}^{N} i - \sum_{i=h}^{N} i = \frac{1}{2}N(N+1) - \frac{1}{2}h(h+1).$$

∎

As a consequence of the above lemma we obtain the following theorem.

Theorem 3.1. *If $k>1$ then the total communication time required to perform any form of Gaussian elimination algorithm on a broadcast bus architecture satisfies:*

$$t_B \geq (\frac{N^2}{2} - \frac{2N^2}{3k})\tau_B.$$

Proof. The function $x \rightarrow x(x+1)/2$ is an increasing function of the variable x when $x \geq 0$ and therefore

$$h(h+1)/2 \leq \frac{1}{2}(\frac{2N}{\sqrt{3k}})(\frac{2N}{\sqrt{3k}} + 1) = \frac{2N^2}{3k} + \frac{N}{\sqrt{3k}}$$

Hence

$$n_B \geq \frac{1}{2}N^2 - \frac{2N^2}{3k} + (\frac{N}{2} - \frac{N}{\sqrt{3k}})$$

When $k > 1$, the term between parentheses in the right hand side of the above expression is positive. This, with the observation made before lemma 3.1 completes the proof.

∎

The theorem shows that with a broadcast bus interconnection network, and for $k \geq 2$, the execution time for Gaussian elimination will be at least quadratic in N, no matter how many processors are used. The minimum of the above bound is reached for $k = 2$ and yields $t_B \geq \frac{1}{6}N^2\tau_B$.

4. Communication complexity for the multiprocessor ring

Following the first method suggested in Section 2.2 for establishing lower bounds on communication complexity, we start by establishing a lower bound for the total number of elementary transfers, i.e. the overall total number of movements of any one element from one processor to a neighbor. The following inequality which is easy to prove by considering the integral of the function x^ν in the interval $[0, N]$, will be quite useful in the remainder of the paper :

$$\forall \nu \geq 0, \quad \sum_{i=1}^{N} i^\nu \geq \frac{N^{\nu+1}}{\nu+1}. \tag{4.1}$$

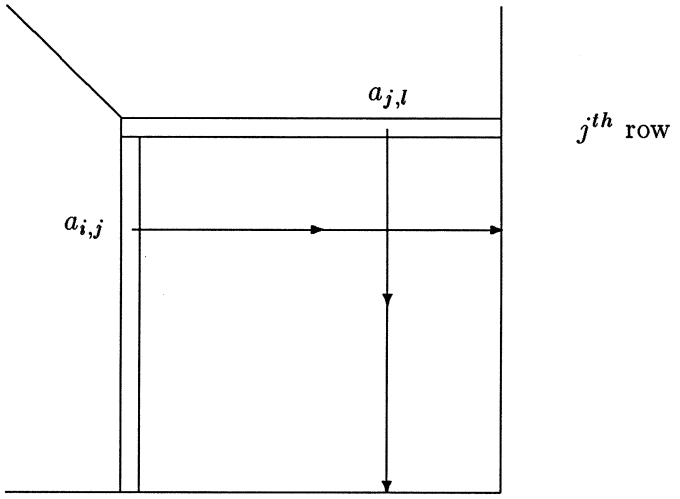

Figure 5: The j^{th} step of Gaussian elimination.

Lemma 4.1. *The minimum number of elementary data transfers required to perform the Gaussian elimination algorithm on a nearest neighbor k-processor ring satisfies:*

$$n_R \geq \frac{1}{16}N^2k - \frac{1}{2}N^2.$$

Proof. Consider the j^{th} step of the algorithm as illustrated in figure 5. Each element $a_{jl}, l = j, j+1, ..N$ must be made available to all processors containing at least one element of the l^{th} column. (We ignore the right hand side f). Although we refer to the j^{th} step of Gaussian elimination here, implying that we consider a lock-step type implementation, it is clear that other forms of Gaussian elimination will involve the same data movement requirement : at some point each of the elements a_{jl} must be made available to any of the processors containing at least one element of the l^{th} column. Let β_l be the minimum number of edges that must be crossed by a_{jl} in order to achieve this. We refer to the corresponding path as the north-south path of a_{jl}. Similarly, let γ_i, $i = j+1, j+2, ..N$, be the length of the path, hereafter referred to as the west-east path of a_{ij}, which a_{ij} must cover in order to become available in each processor holding any part of the i^{th} row. We would like to find a lower bound for the sum

$$n_j = \sum_{i=j+1}^{N} \gamma_i + \sum_{l=j}^{N} \beta_l. \tag{4.2}$$

Observe that if $M \equiv (N-j+1)$ and $s \equiv N^2/k$ then the M x M matrix $\{a_{i,l}, l = j, N; i = j, N\}$, is contained in a total of at least $k_1 \equiv \lceil M^2/s \rceil$ different processors. Consider first the element $a_{j,j}$ $(l = j)$, which is in some processor, say P_1, and let P_D be the processor among the above group of k_1 processors, which is the farthest away from P_1.

For the ring it is easily seen that given an arbitrary group of μ processors, any member of this group admits at least one processor which is at a distance no less than

$$\delta_R(\mu) = \lceil (\mu - 1)/2 \rceil. \tag{4.3}$$

The function δ_R is the reciprocal of the function σ defined by Gentleman [3].

An important observation is that any of the processors of the group of k_1 processors can be reached by $a_{j,j}$ by following part of its path north-south and part of a path west-east of some row i_1. As a result it is possible for $a_{j,j}$ to reach P_D by crossing at most $\beta_j + \gamma_{i_1}$ edges. Hence

$$\beta_j + \gamma_{i_1} \geq \delta_R(k_1). \tag{4.4}$$

Next consider the element $a_{j,j+1}, (l = j+1)$ and the $(M-1)$ x M matrix obtained by deleting the i_1^{th} row. This matrix is contained in at least $k_2 \equiv \lceil (M-1)M/s \rceil$ different processors, and by the same argument as above there exists at least one processor which is at a distance $\delta_R(k_2)$ from that containing $a_{j,j+1}$. Furthermore, this processor can be reached by $a_{j,j+1}$ by crossing part of its path north-south and part of the path west-east of some $a_{i_2,j}$ and we have again $\beta_{j+1} + \gamma_{i_2} \geq \delta_R(k_2)$. This argument can be repeated for $l = j+3, \ldots N-1$, with at each time an inequality of the form

$$\beta_l + \gamma_{i_h} \geq \delta_R(k_h), \tag{4.5}$$

where we have set $h \equiv l - j$ for convenience and where

$$k_h \equiv \lceil (M - h + 1)M/s \rceil. \tag{4.6}$$

Since the rows i_h are deleted as soon as their γ_{i_h} are used, each row index i will appear once and only once, i.e. each γ_i in (4.2) and each $\beta_l, j \leq l \leq N-1$ is accounted for once and only once. For the term β_N in (4.2), which is not yet accounted for, we can use a separate argument. The N^{th} column in contained in at least $k_M = \lceil M/s \rceil$ processors. Note that this is also obtained by letting $h = M$ in (4.6). Hence the path north-south of $a_{j,N}$ satisfies

$$\beta_N \geq \delta_R(k_M). \tag{4.7}$$

Adding the inequalities (4.5) for $l = j, N-1$ and (4.7) we get

$$n_j = \sum_{l=j}^{N-1} (\beta_l + \gamma_{i_h}) + \beta_N \geq \sum_{h=1}^{M} \delta_R(k_h) \tag{4.8}$$

$$\begin{aligned} n_j &\geq \sum_{h=1}^{M} \lceil (k_h - 1)/2 \rceil \geq \sum_{h=1}^{M} (k_h - 1)/2 \\ &\geq \sum_{h=1}^{M} \frac{1}{2} \left(\frac{M}{s}(M - h + 1) - 1 \right) \\ &\geq \frac{1}{4s} M^3 - \frac{1}{2} M \end{aligned}$$

(Note that for step $j = N$ there is no data transfer and $n_N = 0$, but the above inequality is valid because its right hand side is negative.) Summing-up the above for $j = 1, .. N$ and using (4.1) we get

$$\sum_{j=1}^{N} n_j \geq \frac{1}{4s} \frac{N^4}{4} - N(N+1)/4.$$

Replacing s by N^2/k and using the inequality

$$\frac{1}{2} N(N+1) \leq N^2 \tag{4.9}$$

yields the result.

∎

As an immediate consequence we have the following corollary which establishes a bandwidth lower bound for either of the two forms of Gaussian elimination.

Corollary 4.1. *The communication time required to perform any form of the Gaussian elimination algorithm on a multiprocessor ring is such that*

$$t_R \geq t_R^b \equiv (1 - \frac{8}{k})\frac{N^2}{16}\tau_R. \tag{4.10}$$

Proof. The result follows by dividing the lower bound for the total number n_R of elementary data movements given by Lemma 4.1, by the total number of links available in a ring, which is k.

∎

Next we derive a lower bound that takes into account latency rather than total bandwidth, i.e. a latency-type lower bound as defined in Section 2.2.

Lemma 4.2. *The minimum communication time required to perform the Gaussian elimination algorithm on a nearest neighbor k-processor ring satisfies:*

$$t_R \geq t_R^l \equiv \frac{1}{12}(k-3)N\tau_R, \tag{4.11}$$

for a lock-step implementation and

$$t_R \geq t_R^l \equiv \frac{1}{2}(k-1)\tau_R, \tag{4.12}$$

for a pipelined implementation.

Proof. In the proof of Lemma 4.1, we have shown that at each step j of Gaussian elimination the element a_{jj} travels a distance β_j while some element $a_{i_1,j}$ travels the distance γ_{i_1} satisfying

$$\beta_j + \gamma_{i_1} \geq \delta_R(k_1) \geq \lceil (k_1 - 1)/2 \rceil \geq (k_1 - 1)/2 \geq (\frac{1}{s}M^2 - 1)/2 \tag{4.13}$$

Hence $max(\beta_j; \gamma_{i_1}) \geq \frac{1}{2}\delta_R(k_1)$ and therefore at each step there is a latency time of at least $\frac{1}{2}(\frac{1}{2s}M^2 - \frac{1}{2})\tau_R$. For the lock-step implementation, we can add these times for $j = 1, ..N$, use (4.1) and replace s by N^2/k to obtain (4.11).

For the pipelined implementations, the communication of the information at different steps of the outer loop of the algorithm can take place at the same time and therefore, we can only take as a lower bound the largest of the times (4.13) for all j, i.e. the one corresponding to $j = 1$ which, by substitution in (4.13) yields the result.

∎

The times t_R^b and t_R^l given by (4.10) and (4.11)-(4.12) are the bandwidth and latency lower bounds respectively. Notice the important fact that for large k, it is the latency times that dominate. Thus for situations where k is small with respect to N, the bandwidth bound is sharper while for large k the latency bound becomes sharper. A convenient way of taking advantage of both situations is by taking the maximum of both lower bounds as is done in the following theorem.

Theorem 4.1. *The minimum communication time required to perform the Gaussian elimination algorithm on a k-processor ring satisfies:*

$$t_R \geq Max\{t_R^b; t_R^l\} \geq \frac{1}{2}\{t_R^b + t_R^l\} = \frac{1}{2}\left(\frac{N^2}{16}(1-\frac{8}{k}) + \frac{kN}{12}(1-\frac{3}{k})\right)\tau_R \tag{4.14}$$

for a lock-step implementation, and

$$t_R \geq Max\{t_R^b; t_R^l\} \geq \frac{1}{2}\{t_R^b + t_R^l\} = \frac{1}{2}\left(\frac{N^2}{16}(1-\frac{8}{k}) + \frac{1}{2}(k-1)\right)\tau_R \tag{4.15}$$

for a pipelined implementation.

As an example when $k = N^2$ the execution time is at least of the order $O(N^3)$ for the lock-step implementation and $O(N^2)$ for more general implementations. Gentleman's result [3] corresponds to the restricted case $k = N^2$ and concerns the general pipelined algorithms. It is based on latency times only. His lower bound for computing the inverse of a matrix is $O(N^2)$, the same as our result, but he does not restrict himself to Gaussian elimination for computing A^{-1}. Note that we do not claim that the above lower bounds can be reached. In fact the proofs indicate that this is unlikely. We are interested only in the highest order terms (with respect to N and k).

5. Communication complexity for the multiprocessor grid

In this section, the k processors under consideration are connected in a $\sqrt{k}$ x $\sqrt{k}$ square grid, as shown in Figure 3, where $\sqrt{k}$ is an integer. We follow the same steps as in the previous section and start by showing the following analogue of Lemma 4.1.

Lemma 5.1. *The minimum number of elementary data transfers required to perform the Gaussian elimination algorithm on a nearest neighbor $\sqrt{k}$ x $\sqrt{k}$ multiprocessor grid satisfies:*

$$n_G \geq \frac{\sqrt{2}}{9}N^2\sqrt{k} - N^2. \tag{5.1}$$

Proof. The only difference with the proof of Lemma 4.1 is that the function δ_R is replaced by a new function δ_G related to the new architecture. For a group of μ different processors on the grid $\delta_G(\mu)$ represents a lower bound for the distance from any member of this group to its most remote processor in the group. A simple calculation shows that

$$\delta_G(\mu) = \lceil \frac{1}{2}(\sqrt{2\mu - 1} - 1) \rceil$$

but for convenience we will use the simpler lower bound

$$\delta_G(\mu) \geq \lceil \sqrt{\frac{\mu}{2}} - 1 \rceil \tag{5.2}.$$

Resuming the proof of 4.1 from (4.8) , we have

$$n_j = \sum_{l=j}^{N-1}(\beta_l + \gamma_{i_h}) + \beta_N \geq \sum_{h=1}^{M}\delta_G(k_h) \tag{5.3}$$

$$n_j \geq \sum_{h=1}^{M} \lceil \sqrt{\frac{k_h}{2}} - 1 \rceil \geq \sum_{h=1}^{M} \left(\sqrt{\frac{k_h}{2}} - 1 \right)$$
$$\geq \sum_{h=1}^{M} \left(\sqrt{\frac{M(M-h+1)}{2s}} - 1 \right)$$
$$\geq \sqrt{\frac{M}{2s}} \frac{M^{3/2}}{3/2} - M = \frac{2}{3\sqrt{2s}} M^2 - M.$$

After summation over $j = 1, ..N$ and use of inequality (4.1) and (4.9) we get the desired result. ∎

In a square $\sqrt{k}$ x $\sqrt{k}$ grid, there is a total of $2\sqrt{k}(\sqrt{k}-1) = 2(k-\sqrt{k})$ links. Therefore, the above lemma leads to the following corollary.

Corollary 5.1. *On a multiprocessor grid the communication time for any form of the Gaussian elimination algorithm satisfies*

$$t_G \geq t_G^b \equiv \left(\frac{1}{9\sqrt{2}} - \frac{1}{2\sqrt{k}} \right) \frac{N^2}{\sqrt{k}-1}. \tag{5.4}$$

Proceeding as in the previous section, we obtain the following latency lower bound.

Lemma 5.2. *The communication time for the Gaussian elimination algorithm, implemented on a* $\sqrt{k}$ x $\sqrt{k}$ *multiprocessor grid satisfies*

$$t_G \geq t_G^l \equiv (\frac{1}{4\sqrt{2}}\sqrt{k}N - \frac{1}{2}N)\tau_R, \tag{5.5}$$

for a lock-step implementation and

$$t_G \geq t_G^l \equiv \frac{1}{2}\left(\sqrt{\frac{k}{2}} - 1 \right) \tau_R, \tag{5.6}$$

for a pipelined implementation.

Proof. The proof is similar to that of Lemma 5.1. We have $\beta_j + \gamma_{i_1} \geq \sqrt{\frac{M^2}{2s}} - 1$ and therefore

$$max\{\beta_j; \gamma_{i_1}\} \geq \frac{1}{2}(\sqrt{\frac{M^2}{2s}} - 1). \tag{5.7}$$

At each step of Gaussian elimination we therefore have a latency time of at least $\frac{1}{2}(\sqrt{\frac{M^2}{2s}} - 1)\tau_G$. The result follows by summing up the times (5.7) over $j = 1, ..N$ gives the result (5.5) while taking their maximum, i.e., the vaule of (5.7) for $j = 1$, yields the result (5.6). ∎

Combining Corollary 5.1 and the above lemma, the following theorem can be derived.

Theorem 5.1. *When $k>1$, the minimum communication time for the Gaussian elimination algorithm implemented on a $\sqrt{k}$ x $\sqrt{k}$ multiprocessor grid satisfies:*

$$t_G \geq Max\{t_G^b; t_G^l\} \geq \frac{1}{2}\{t_G^b + t_G^l\} \geq \frac{1}{2}\left(\frac{N^2}{9\sqrt{2k}} + \frac{\sqrt{k}N}{4\sqrt{2}} - (\frac{N^2}{k} + \frac{1}{2})\right)\tau_G \tag{5.8}$$

for a lock-step implementation and

$$t_G \geq Max\{t_G^b; t_G^l\} \geq \frac{1}{2}\{t_G^b + t_G^l\} \geq \frac{1}{2}\left(\frac{N^2}{9\sqrt{2k}} - \frac{N^2}{k} + \frac{1}{2}(\sqrt{\frac{k}{2}} - 1)\right)\tau_G \tag{5.9}$$

for a pipelined implementation.

Proof. Here we have used the inequalities $\sqrt{k} - 1 \leq \sqrt{k}$ and $k - \sqrt{k} \geq \frac{1}{2}k$, (which is true when $\sqrt{k} \geq 2$ i.e. when $k>1$), to simplify the right hand side of (5.4). ∎

The above lower bounds start by decreasing when $\sqrt{k}$ increases from 2 and then may increase again as k becomes large. This shows that it is not always possible to achieve ever higher speed-up by using more processors, even though the arithmetic time might be reduced by factors as high as N^2 using $k = N^2$ processors. For the pipelined methods for example, the lower bound for communication cannot be less than $O(N)$ since the bandwidth term is at least of the order of $O(N)$ because $k \leq N$. This means that communication has the same order of complexity as the*best* possibele arithmetic complexity. This tells us that in machines where communication channels are relatively slow, any type of Gaussian elimination will be communication bound.

Let us now consider in some detail the lock-step methods. Let ω be the time to perform a pair of arithmetic operations consisting of an addition and a multiplication. The arithmetic operations in the Gaussian elimination algorithm will consume a time of $\frac{N^3}{3k}\omega$ at best. Adding this to the lower bound (5.8) we get the minimal time in which a lock-step Gaussian elimination can be executed on a $\sqrt{k}$ x $\sqrt{k}$ grid. Neglecting the low order terms the resulting total time is of the form

$$\alpha_1\frac{N^3}{k}\omega + (\alpha_2\frac{N^2}{\sqrt{k}} + \alpha_3 N\sqrt{k})\tau_R, \tag{5.10}$$

where $\alpha_i, i = 1,3$ are some constants. One can now minimize the above function with respect to the number of processors k. With the correct coefficients $\alpha_i, i = 1,3$ the resulting expressions are complicated and difficult to interpret. However, in the next section we will propose an example of a lock-step implementation of the Gaussian elimination algorithm which leads to a timing formula of the form (5.10) with simpler coefficients. As will be seen, the minimum of the above function is of the order $O(N^{5/3})$. This is not a big improvement over the optimal time $O(N^2)$ obtained for the ring. Thus one conclusion to draw is that lock-step implementations have a limited speed-up for large number of processors.

6. A Lock-Step Gaussian Elimination Algorithm for the Multiprocessor Grid

For the purpose of illustrating our lower bounds, we will describe in this section an example of a lock-step implementation of the Gaussian elimination algorithm on a square multiprocessor grid. Consider the simplest distribution of the data among the k processors consisting in mapping the matrix naturally onto the array. The matrix is partitioned into k square blocks of size $(N/\sqrt{k})$ x $(N/\sqrt{k})$ each, while the right hand side is partitioned into $\sqrt{k}$ equal parts which are

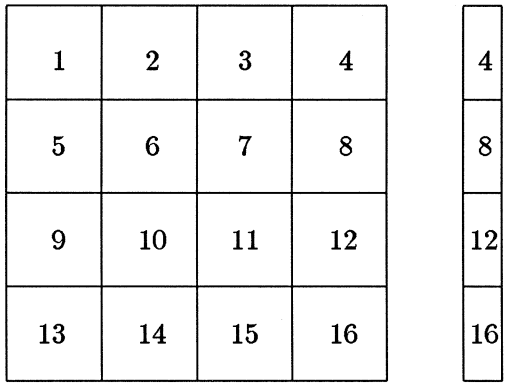

Figure 6: Distribution of the linear system in $k = 16$ processors.

distributed into the processors of the last column of the grid. The situation is illutrated in Figure 6, where the numbers in each block indicate the processor to which the block are assigned.

We denote by m the block-size of A, i.e. $m = \frac{N}{\sqrt{k}}$. Consider now the time it takes to execute the lock-step Gaussian elimination algorithm with no pivoting, on such a grid. Each step of the algorithm requires that we move the pivoting row, i.e. row j, so that the processors under those containing this row will be able to perform the eliminations. The j^{th} row whose length is $N-j+1$, consists of $\lceil (N-j+1)/m \rceil$ blocks of size m each, (except the first block whose length is $(N-j+1)$ Mod m), with each block in one processor. Each of these processors, must transfer its block downwards to all processors below it.

At the same time the processors containing the blocks of the j^{th} column must transfer the pivots to all processors to their right. Since the transfer in the horizontal and vertical directions can be overlapped, the partial transfer time t_j at step j will be the same as that of transferring one block of m elements from one processor to its $\lceil (N-j+1)/m \rceil$ immediate neighbors, i.e.

$$t_j = (\lceil (N-j+1)/m \rceil + m)\tau_G$$

In order to simplify the summation of the above partial times through $j = 1, N-1$, we will approximate the above time by

$$t_j \approx \left(\frac{N-j}{m} + m \right) \tau_G.$$

Summing up over the steps $j = 1, ..N-1$ we find that

$$t_G \approx \left(\frac{N^2}{\sqrt{k}} + \frac{1}{2}\sqrt{k}N \right) \tau_G. \tag{6.1}$$

Once the transfers of the necessary parts of the j^{th} row and column are completed, each of the active processors will perform m eliminations each at the cost of $m^2\omega$, where ω is the time to perform one multiplication and one addition. Therefore, the total time for performing the arithmetic operations is

$$t_{a,G} \approx \frac{N^3}{k}\omega. \tag{6.2}$$

Adding (6.1) and (6.2) we find the approximate total run time

$$t \approx \frac{N^3}{k}\omega + \left(\frac{N^2}{\sqrt{k}} + \frac{1}{2}\sqrt{k}N\right)\tau_G. \tag{6.3}$$

Observe that the above total time is of the same form as (5.10). It is the sum of an increasing and a decreasing function of k and, in general, as k varies from $k = 1$ to $k = N^2$, the execution time of the algorithm starts by decreasing and then increases again. We say "in general" because this may not be true for some particular values of ω and τ_G. For example if τ_G is zero, or if it is very small, then the above total time will always decrease when k increases from 1 to N^2. For $k = N^2$ the time is of the form $O(N^2)$ which means that when k is much larger than N, it may be the case that using less than the total number processors will be better than using all of them.

This raises the question of optimality: what is the number of processors k_{opt} that achieves the smallest possible execution time t_{opt} ? Let us call $t(k)$ the approximate total execution time provided by the right hand side of (6.3), i.e.

$$t(k) \equiv \frac{N^3}{k}\omega + \frac{1}{2}\sqrt{k}N\tau_G + \frac{N^2}{\sqrt{k}}\tau_G. \tag{6.4}$$

Differentiating (6.4) with respect to k leads to a third degree equation whose explicit solution is rather complicated. Fortunately, we can find a nearly optimal solution, by simply considering the first two terms of $t(k)$ as defined by (6.4). For all k we have:

$$t(k) \geq \frac{N^3}{k}\omega + \frac{1}{2}\sqrt{k}N\tau_G = N^3\tau_G\left(\frac{\rho}{k} + \frac{1}{2}\frac{\sqrt{k}}{N^2}\right) \tag{6.5}$$

where ρ is the ratio of arithmetic time over transfer time, i.e. $\rho = \omega/\tau_G$. The minimum of the right hand side of (6.5) is achieved for

$$k_* = (4N^2\rho)^{2/3}, \tag{6.6}$$

which is valid when $1 \leq k_* \leq N^2$, i.e. when $\frac{1}{4N^2}\rho \leq N/4$. Replacing (6.6) in (6.5) yields the inequality

$$\forall k, \quad t(k) \geq t_* \equiv \frac{3\omega}{(4\rho)^{2/3}}N^{5/3}. \tag{6.7}$$

Let us now show that the time t_* is nearly optimal. From the definition of $t(k)$ we have

$$t_{opt} \leq t(k_*) = t_* + \frac{\tau_G}{(4\rho)^{1/3}}N^{4/3},$$

where we have used (6.6) and (6.4). By (6.7), $t_{opt} \geq t_*$ and hence

$$0 \leq t_{opt} - t_* \leq \frac{\tau_G}{(4\rho)^{2/3}}N^{4/3}.$$

or

$$0 \leq \frac{t_{opt} - t_*}{t_*} \leq (4\rho^4)^{1/3}N^{-1/3},$$

which means that t_* is a good approximation to t_{opt}, when N is large.

Note that by the similarity of (6.4) and (5.10), we have also proved that

Theorem 6.1. *The minimum time for performing the lock-step Gaussian algorithm on a multiprocessor grid with an arbitrarily large number of processors is asymptotically of the form* $O(N^{5/3})$.

The above result is disappointing in that the optimal time $O(N^{5/3})$ can be regarded as a poor improvement over the $O(N^2)$ time of the ring, considering the additional hardware complexity involved. The explanation for this ineffiency lies in the fact that when k increases, one must send data to processors that become farther and farther apart from each other. The distance behaves like $O(\sqrt{k})$ and appears as an incresing function in the communication time.

A hardware solution for avoiding the difficulty is to provide for a broadcast facility. Suppose for example that each vertical path between the processors on the same vertical line, is a bus that allows for broadcasting and similarly, each horizontal line is a broadcast bus. For such busses, the time to send a vector of length m from one processor to some other processors is of the form $m\tau_B$, independent of the number of processors to which the data is sent. Thus, with this new feature, the latency times are inexistant and the communication time becomes

$$\frac{N^2}{\sqrt{k}}\tau_B,$$

which is always a decreasing function of k. Since $k \leq N^2$ this would lead us to an achievable linear time for the lock-step Gaussian elimination, just as for general pipelined algorithms.

It may then appear that a general remedy to the latency problem for grid connected machines would be to link the processors in each of the horizontal and vertical lines of the grid by a broadcast bus. However, this is true for an algorithm such as the lock-step Gaussian elimination which requires mainly broadcast type data transfers, but may not be of any help for other algorithms which require more local data interchange. For example, in the solution of triangular systems, some methods move data packets of equal size simultaneously from a number of processors to their immediate neighbors [5]. In those cases the bus must be time-shared by the processors which effectively reduces its actual speed. For these algorithms, the bus solution does not appear to be a good choice. The approach of combining local links and more global communication is reasonable when the number of processors is large and has been adopted in several parallel architecture projects, e.g. the Finite Element Machine [1], the CEDAR machine [6].

7. Conclusion

The analysis of the Gaussian Elimination algorithm proposed in this paper shows that communication complexity may be a serious obstacle to speed-up in parallel computation. Considering arithmetic alone one might believe that by using $k = N^2$ processors is is always possible to speed-up Gaussian elimination by an order of N^2, thus leading to an algorithm whose run time is linear in N. We have shown that for a bus or a ring architecture this is impossible.

Moreover, for a grid architecture it has been shown that lock-step implementations of Gaussian elimination whereby the successive steps of the outer loop do not overlap, are not competitive for large number of processors: a linear time can be achieved for pipelined implementation while only a time of the order of $O(N^{5/3})$ is possible for the lock-step methods. However, for small number of processors lock-step methods may perform quite well. In a related topics, Saad and Schultz [11] have shown that for very large number of processors a (pipelined) wavefront implementation of Gaussian elimination for solving *banded* linear systems is best but that for small number of processors a straight-forward lock-step Gaussian elimination outperforms the wavefront approach.

Gaussian elimination is a communication intensive process and the main difficulty in attempting to speed-up the algorithm stems from the fact that increasing the number of processors also

increases the distance between the data in nearest-neighbor type architectures. Many numerical methods, such as the solution of partial differential equations by finite difference/finite element methods, can be formulated so as to require local communication only. For such problems, architectures based on nearest neighbor interconnection are perfectly suitable. However, for communication intensive problems, such as the solution of dense linear systems considered in this paper, nearest neighbor architectures may seriously impede performance if the algorithm is not carefully designed in account of communication delays.

Acknowledgement.

The author is indebted to Dr. Ilse Ipsen for making helpful comments and pointing out an error in an early version of this paper.

References

[1] L.M. Adams, *Iterative algorithms for large sparse linear systems on parallel computers,* Ph.D. Thesis, University of Virginia, Applied Mathematics, 1982. Also available as NASA Contractor Report# 166027.

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

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

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

[5] I. Ipsen, Y. Saad, M.H. Schultz, *Complexity of dense linear system solution on a multiprocessor ring,* Technical Report 349, Computer Science Dept., Yale University, 1984.

[6] C. Kamath and A. Sameh, *The preconditioned conjugate gradient algorithm on a multiprocessor,* Technical Report ANL/MCS-TM-28, Argonne National Lab., 1984.

[7] 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.

[8] C.A. Mead,L.A. Conway, *Introduction to VLSI Systems,* Addison Wesley, Mass.,USA, 1980.

[9] 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.

[10] D.A. Reed, M.L. Patrick, *Parallel, Iterative Solution of Sparse Linear Systems : Models and Architectures,* Technical Report 84-35, ICASE, 1984.

[11] Y. Saad, M.H. Schultz, *Direct parallel methods for solving banded linear systems,* Technical Report YALEU/DCS/RR-387, Computer Science Dept., Yale University, 1985.

[12] 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.