A Note on Gaussian Elimination with Partial Pivoting on an MIMD Computer

Marinus Veldhorst

RUU-CS-84-14
December 1984

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

A Note on Gaussian Elimination with Partial Pivoting on an MIMD Computer

Marinus Veldhorst

Technical Report RUU-CS-84-14
December 1984

Department of Computer Science
University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht
the Netherlands

A Note on Gaussian Elimination with Partial Pivoting on an MIMD Computer

Marinus Veldhorst

Department of Computer Science, University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht, The Netherlands.

ABSTRACT

A parallel algorithm for an MIMD computer will be presented that runs in time n^2-1 and needs 0.3536..n processors in order to perform a Gaussian elimination with partial pivoting on an n×n matrix.

Keywords: Numerical linear algebra, parallel algorithms, Gaussian elimination, MIMD computer.

1. Introduction

The problem of solving a system of linear equations on an MIMD computer has been dealt with by R.S. Lord, J.S. Kowalik and S.P. Kumar (cf. [1]). They solved the problem with a special selection of tasks of Gaussian elimination with partial pivoting (see Figure 1). This selection led to a precedence graph for the set of tasks $J = \{T_i^j : 1 \le i \le n-1, i \le j \le n\}$ (see Figure 2). The precedence relation << is defined as

$$T_i^j << T_m^k \quad \text{iff} \quad j<k \quad i=m, \quad i<m. \tag{1}$$

If $T_i^j << T_m^k$ the execution of task T_m^k is not allowed to start before the execution of T_i^j is finished. The authors assigned to each task a weight W that denotes the number of time steps required for the execution of this task. They considered one time step to consist of one multiply and

```
Program LUDECOMP(A(n,n))
  for k := 1 to n-1 do
    Find p such that                                         ⎫
    |A(p,k)| = max(|A(k,k)|,...,|A(n,k)|)                    ⎪
    PIV(k) := p {pivot row}                                  ⎪
    interchange A(PIV(k),k) and A(k,k)                       ⎬ T^k_k
    c := 1/A(k,k)                                            ⎪
    for i := k+1 to n do                                     ⎪
      A(i,k) := A(i,k)*c      {elements of L}                ⎭
    for j := k+1 to n do                                     ⎫
      interchange A(PIV(k),j) and A(k,j)                     ⎬ T^j_k, j>k
      for i := k+1 to n do                                   ⎪
        A(i,j) := A(i,j) - A(i,k)*A(k,j)                     ⎭
```

Fig. 1. Program for LU decomposition with illustration of tasks.

one subtraction or one multiply and one compare. Thus they ignored any overhead for loop control. This assigned the following weights to tasks:

$$W(T_i^j) = \begin{cases} n+1-i & \text{if } i=j \\ n-i & \text{if } i<j \end{cases} \qquad (2)$$

In this way the precedence graph with the weights becomes a weighted graph. They observed that the longest path consists of the tasks

$$T_1^1,\ T_1^2,\ T_2^2,\ T_2^3,\ \ldots,\ T_{n-1}^{n-1},\ T_{n-1}^{n}.$$

Any scheduling of the tasks on several processors will therefore require at least

$$\sum_{i=1}^{n-1} \left(W(T_i^i) + W(T_i^{i+1}) \right) = n^2-1 \qquad (3)$$

time.

The authors of [1] specified a schedule of these tasks on $\lceil n/2 \rceil$ processors such that these processors execute the task system in time n^2-1. With this result they obtained an efficiency E_p of 2/3 for $p=\lceil n/2 \rceil$ processors.

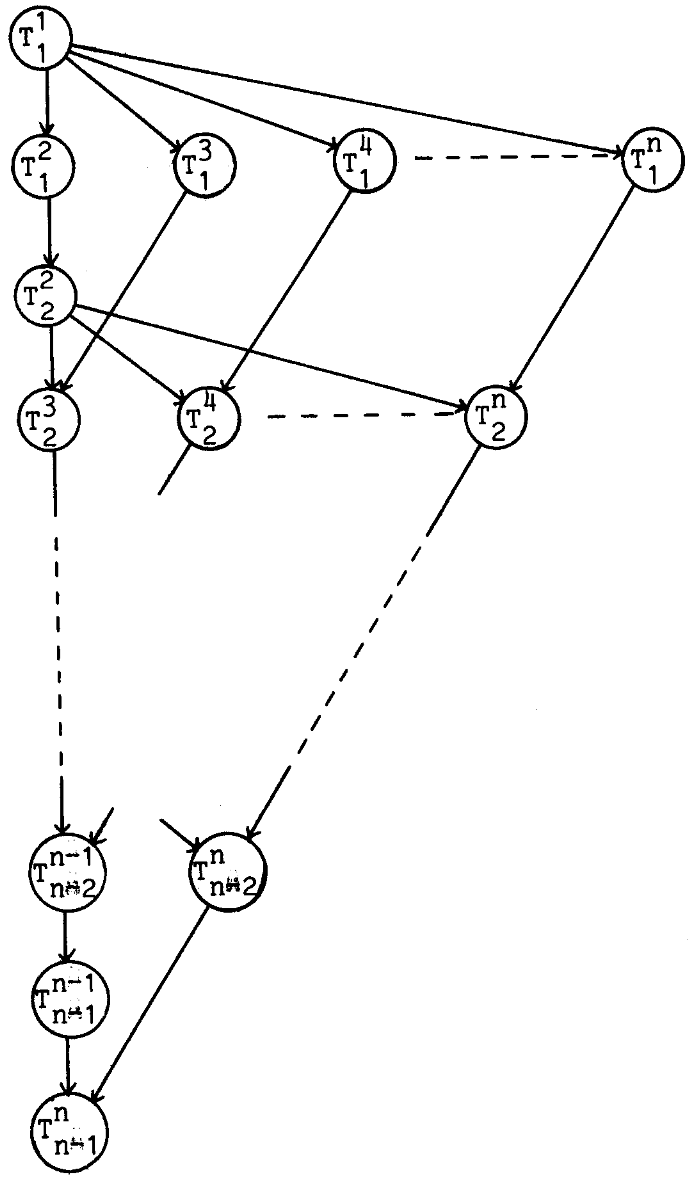

Fig. 2. Precedence graph for task system given in Fig. 1.

$E_p = \lim_{n\to\infty} S_p/p = 2/3$ in which

$S_p = t_1/t_p$ is the speed up and

t_i is the execution time when i processors are used.

They derived also an asymptotic lower bound of

$$\alpha n \text{ processors } (\alpha=0.34729\ldots) \tag{4}$$

that can execute the task system in time n^2-1 (α is a solution of the equation $3\alpha-\alpha^3=1$). This lower bound on the number of processors gives an upper bound on the efficiency of

$$1/3\alpha = 0.9589\ldots \text{ for large } n.$$

In the next section we will derive a smaller number of processors that can execute the task system J in the same time n^2-1. We will prove that βn processors will do the job for each β with

$$\beta n \geq \frac{1+\sqrt{2n^2+2n+1}}{4}.$$

Thus β is rather close to the lower bound (4) and the corresponding asymptotic efficiency for the smallest β will be

$$E_{\beta n} = S_{\beta n}/\beta n = 0.9428..$$

2. A more efficient scheduling

In this section we will present a scheduling of the task system J on p processors $P_1,\ldots,P_p$ ($p\geq(1+\sqrt{2n^2+2n+1})/4$) such that the tasks of J can be executed in time n^2-1. Let us define the following sequences of tasks

$$r_1 = \{T_1^1;\ T_1^2;\ T_2^2;\ T_2^3;\ \ldots;\ T_{n-2}^{n-1};\ T_{n-1}^{n-1};\ T_{n-1}^{n}\} \tag{5}$$

$$r_j = \{T_1^j;\ T_2^j;\ \ldots;\ T_{j-2}^j\} \quad 3\leq j\leq n.$$

Observe that (i) the tasks in one r_j must be executed in the order as they are enumerated in (5) and (ii) that no task in r_i ($i\geq3$) is a direct predecessor of any task in r_j ($j\geq3$) in the precedence graph. We will use the term r-sequence to denote a sequence r_j of tasks for some j. With the weight $W(r_j)$ we denote the sum of the weights of the tasks of

r_j.

Now we will look for a scheduling of tasks such that all tasks of one r-sequence are to be executed by one processor. Thus, the problem of scheduling the task system is split into two problems:

(i) How to assign r-sequences to processors?

(ii) How to schedule for each processor P_j the tasks of the r-sequences assigned to P_j?

The assignment and scheduling chosen should allow all tasks to be executed in time n^2-1 satisfying all precedence constraints.

<u>Assignment of r-sequences</u>.

P_1 executes all tasks of r_1,

P_{p-j} executes the tasks of r_{n-j}, $r_{n-2p+3+j}$ and $r_{n-2p+2-j}$ if $0 \le j \le n-2p-1$,

P_{p-j} executes the tasks of r_{n-j} and $r_{n-2p+3+j}$ if $n-2p-1<j\le p-2$.

<u>Proposition 1</u>. With $p \ge (n+1)/3$ each r-sequence is assigned to some processor.

<u>Example</u>. With n=16 and p=7 we obtain the assignment

P_1 executes all tasks of r_1,

P_2 executes all tasks of r_{11} and r_{10},

P_3 executes all tasks of r_{12} and r_9,

P_4 executes all tasks of r_{13} and r_8,

P_5 executes all tasks of r_{14} and r_7,

P_6 executes all tasks of r_{15}, r_6 and r_3.

P_7 executes all tasks of r_{16}, r_5 and r_4.

The r-sequences $r_3,\dots,r_{16}$ are assigned to processors $P_2,\dots,P_7$ in a snake-like way.

Let us now consider the scheduling of tasks on one processor. If we do not want P_1 to wait, then there must be deadlines for tasks of r_j ($j \ge 3$) that precede directly some task of r_1. Thus we have deadlines d_j for each r_j:

$$d_j = n + 2\sum_{h=1}^{j-2}(n-h) = n(2j-3)-(j-2)(j-1). \qquad (6)$$

On the other hand a task T_i^j cannot start before task T_i^i is finished. Thus there is a starting time s for each task:

$$s(T_i^j) = 1+n + 2\sum_{h=1}^{i-1}(n-h) = 2ni-i^2-n+i+1$$

Whether a process T_i^j can really start on its starting time (i.e., all processes $T_k^m \ll T_i^j$ have finished then) depends on it whether P_1 had to wait or not.

Let us now give the scheduling of tasks on one processor. For this we distinguish between processors that have been assigned two r-sequences and processors that have been assigned three r-sequences. As already observed in [1] the scheduling of the tasks of two r-sequences on one processor is not difficult.

<u>Schedule</u> <u>S</u>.

(1) If $n-2p-1<j\leq p-2$ the tasks of $r_{n-2p+3+j}$ and r_{n-j} are scheduled in the order (with $m=n-2p+3+j$ and $q=n-j$) on processor P_{p-j}

$$T_1^m,\ T_1^q,\ T_2^m,\ T_2^q,\ \ldots,\ T_{m-2}^m,\ T_{m-2}^q,\ T_{m-1}^q,\ \ldots,\ T_{q-2}^q.$$

(2) If $0\leq j\leq n-2p-1$ the tasks of $r_{n-2p+2-j}$, $r_{n-2p+3+j}$ and r_{n-j} are scheduled in the order (with $k=n-2p+2-j$, $m=n-2p+3+j$ and $q=n-j$) on processor P_{p-j}

$$T_1^k,\ T_1^m,\ T_2^k,\ T_2^m,\ \ldots,\ T_{k-2}^k,\ T_{k-2}^m,$$
$$T_{k-1}^m,\ T_1^q,\ T_k^m,\ T_2^q,\ \ldots,\ T_{x+k-3}^m,\ T_x^q,$$
$$T_{x+k-2}^m,\ T_{x+k-1}^m,\ \ldots,\ T_{m-2}^m,\ T_{x+1}^q,\ \ldots,\ T_{q-2}^q$$

in which x is the largest integer such that the deadline d_m for r_m is met.

The only thing that remains to prove, is that this schedule allows the task system to be executed in time n^2-1.

<u>Proposition 2</u>. With $n-2p-1<j\le p-2$, Schedule S satisfies the starting times and deadlines.

<u>Proof</u>. Actually this is already pointed out in [1]. With $m=n-2p+3+j$ and $q=n-j$ processor P_{p-j} executes T_i^m and T_i^q during the time that P_1 executes T_i^{i+1} and T_{i+1}^{i+1} $(1\le i\le m-2)$. P_{p-j} will be idle only during time steps 1 to n. Thus execution of T_i^m will be finished after

$$n + W(r_m) + \sum_{i=1}^{m-2} W(T_i^q) = n + 2\sum_{i=1}^{m-2}(n-i)$$

time steps, which satisfies the deadline. For $i>m-1$, P_{p-j} has to execute tasks of r_q only and it will start a task as soon as its starting time has arrived. Obviously, the deadline for r_q will be met.

Q.E.D.

<u>Proposition 3</u>. With $0\le j\le n-2p-1$ and $p\ge(1+\sqrt{2n^2+2n+1})/4$ Schedule S satisfies:

$$m-4 + n + W(r_k) + W(r_m) + W(r_q) \le d_q.$$

<u>Proof</u>.

$$8p^2 - 4p - n^2 - n \ge 0 \quad \text{for all} \quad p\ge(1+\sqrt{2n^2+2n+1})/4.$$

And thus

$$n^2-8p^2+4p-j^2+n+3j-2 \le 0 \quad \text{for all } j \quad \text{for all } p\ge(1+\sqrt{2n^2+2n+1})/4. \qquad (7)$$

Expressing $W(r_k)$, $W(r_m)$, $W(r_q)$ and d_q (with $k=n-2p+2-j$, $m=n-2p+3+j$ and $q=n-j$), in terms of only n, p and j, yields

$$m-4 + n + W(r_k) + W(r_m) + W(r_q) - d_q = (n^2-8p^2+4p+n-j^2+3j)/2$$

With (7) the Proposition holds.

Q.E.D.

This means that in Schedule S we are allowed to have m-4 idle time for processor P_{p-j} ($0 \le j \le n-2p-1$). Maybe there is even more idle time available. In Schedule S it is not explicitly stated where the idle time occurs, but it certainly will not occur before T^m_{k-1} finishes.

<u>Proposition</u> <u>4</u>. With $0 \le j \le n-2p-1$ and $p \ge (1+\sqrt{2n^2+2n+1})/4$, Schedule S satisfies all starting times.

<u>Proof</u>. Starting times and deadlines of T^k_1, T^m_1, ..., T^m_{k-2}, T^m_{k-1} are certainly met. Because $W(T^q_1) > W(T^k_k)$, the starting time of T^m_k is also met. Thus, starting times of all tasks in Schedule S until T^m_{x+k-1} are met and there is no idle time between the execution of these tasks.

There are two cases:

1. The execution of T^m_{x+k-2}, ..., T^m_{m-2} does not introduce idle time. Then starting time and deadline of T^m_{m-2} are met by definition of x. But then the starting time of T^m_{m-3} is also met, etc. The starting times of all tasks of the schedule until T^m_{m-2} are met without idle time. Then obviously the schedule works.
2. The execution of T^m_{x+k-2}, ..., T^m_{m-2} requires some idle time I_t. Without loss of generality we can assume that I_t is concentrated just after T^m_{x+k-1}. This means that the execution of T^m_{m-2} starts exactly on time $s(T^m_{m-2})$. Thus T^m_{m-2} finishes $d_m - s(T^m_{m-2}) + 1 - W(T^m_{m-2})$ time steps before d_m.

   Inserting T^q_{x+1} just after T^m_{x+k-1} (absorbing I_t) would violate the deadline of T^m_{m-2} (by definition of x). Thus

$$I_t + d_m - s(T^m_{m-2}) + 1 - W(T^m_{m-2}) < W(T^q_{x+1}).$$

Hence

$$I_t < n-x-1-n+m-2 = m-x-3.$$

and

$$I_t \leq m-x-4 \leq m-4$$

We were allowed to have m-4 idle time (cf. Proposition 3). Thus, if idle time I_t is required, then it is less than what is available. Hence Schedule S works in time n^2-1.

Q.E.D.

<u>Theorem</u>. With $p \geq (1+\sqrt{2n^2+2n+1})/4$ the task system J can be executed on an MIMD computer with p processors in time n^2-1.

<u>Proof</u>. Follows from Propositions 2, 3 and 4.

Q.E.D.

Schedule S is not given precisely enough to be transformed into a program: though x is well defined, it is not given as a formula. For each processor P_{p-j} $(0 \leq j \leq n-2p-1)$ its x is defined as the largest integer such that the deadline d_m of r_m is met. Hence x is the largest integer such that

$$n + W(r_k) + W(r_m) + \sum_{i=1}^{x} W(T_i^q) \leq d_m$$

with $k=n-2p+2-j$, $m=n-2p+3+j$ and $q=n-j$. This means that x is the largest integer $(0 \leq x \leq n-j-2)$ such that

$$x^2 - (2n-1)x + 8pj-4j+4p-2 \geq 0 \tag{8}$$

Knowing that the left hand side of (8) is monotone decreasing in x on the segment [0,n-1], a binary search can be used to determine the largest x that satisfies (8). This takes O(log n) time and can be done on each processor P_{p-j} $(0 \leq j \leq n-2p-1)$ in the time that P_1 executes T_1^1.

References.

[1] LORD, R.E., J.S. KOWALIK and S.P. KUMAR, Solving Linear Algebraic Equations on an MIMD Computer, J. ACM 30 (1983), pp. 103-117.