NASA Technical Memorandum 103229
ICOMP-90-18

Parallel/Distributed Direct Method for Solving Linear Systems

Avi Lin
Temple University
Philadelphia, Pennsylvania

and Institute for Computational Mechanics in Propulsion
Lewis Research Center
Cleveland, Ohio

July 1990

NASA

INSTITUTE FOR COMPUTATIONAL MECHANICS IN PROPULSION
NASA LEWIS RESEARCH CENTER
ICOMP
CASE WESTERN RESERVE UNIVERSITY

Parallel / Distributed Direct Method for Solving Linear Systems

Avi Lin
Department of Mathematics
Temple University
Philadelphia, Pennsylvania 19122

and Institute for Computational Mechanics in Propulsion*
Lewis Research Center
Cleveland, Ohio 44135

Abstract

In this paper, a new family of parallel schemes for directly solving linear systems will be presented and analyzed. It is shown that these schemes exhibit a near optimal performance and enjoy several important features:

1. For large enough linear systems, the design of the appropriate parallel algorithm is insensitive to the number of processors as its performance grows monotonically with them.
2. It is especially good for large matrices, with dimensions large relative to the number of processors in the system. In this case, it achieves
   - an optimal speed up.
   - an optimal efficiency.
   - a very low communication time complexity.
3. It can be used in both distributed parallel computing environments and tightly coupled parallel computing systems.
4. This set of algorithms can be mapped onto any parallel architecture without any major programming difficulties or algorithmical changes.

*Work funded by NASA Space Act Agreement C-99066-G.

1 Introduction

Solving a linear system $Ax = b$ for $x \in R^n$, where $A \in R^n \times R^n$ is an $n \times n$ square matrix and $b \in R^n$ is a known vector, is a very basic computing operation in many scientific fields. It is known that the sequential time complexity of a direct procedure for computing x is $O(n^3)$. This time is intolerable when n is large. Therefore a practical parallel algorithm to speed up the computation is almost a must. *Dongarra*, *Gustavson* and *Karp* [1] have analyzed various ways for implementing Gaussian elimination (GE) with implications for several architectures. They have pertubed the three sequential nested loops into six different loop variations. *George*, *Heath* and *Liu* [2] the later formed these six methods for self-scheduling a pool of tasks (for the Cholesky algorithm in particular). The "submatrix" version is the closest method to the algorithms that will be presented in this paper, as they have a "wave-front" like structure. Recently, *Lin* and *Zhang* [3] and *Zhang* and *Lin* [4] have presented a class of efficient parallel algorithms for solving a linear triangular system and parallel algorithms for the LU decomposition of a matrix, which can be applied easily on any parallel machine or distributed environment. Fortunately, it turns out that these algorithms are part of a more general family of algorithms to solve a linear system of equations. Using somewhat similar concepts, an efficient and practical parallel algorithm for solving such a system will be developed in the present paper. It is obvious that by combining these two algorithms, it is possible to get a parallel algorithm for solving the linear system $Ax = b$. However, although this method can be programmed and handled fairly easy, this algorithm turns to be not the best one in this family of parallel algorithms, as there exist algorithms that mix these two steps, resulting in a larger speed-up. There are two important factors needed to be considered when trying to design a useful parallel algorithm: one is communication and the other is the processor waiting time. The communication time is defined to be the time spent on transferring information between the processors or a processor and a memory. The waiting time is defined to be the processor idle time while waiting for the necessary information elements from the other processors. The main goal in designing the parallel algorithm is two-fold:

- to minimize the above two parts of the computing time while using a small amount of processors
- the number of processors should not depend on the input size n.

In the rest of the paper, it will be assumed that the number of available processing elements (PE) in the parallel system is $P+1$, which are indicated by PE_0, PE_1, $\cdots$, PE_P. The topology of the parallel model consists of P PEs connected in a ring like shape, i.e. PE_i is connected to the PE_{i-1} and PE_{i+1} for all $1 \leq i < P$, where PE_0 serves as a master processor as it may have a direct connection to each of the processors in the system. The later connection is not a must for the present analysis but it is convenient for understanding the algorithm. The computational model of the parallel machine is of the loosely coupled system where all the information is passed via the interconnection network and a shared memory is used very seldom. Thus, for the shared memory, one can assumes whatever reasonable model (coomon bus or a special private connection). Each PE can be:

- a regular scalar processor.
- a vector processor.
- another (smaller) parallel computer.

2 The Parallel Algorithm Setup.

Generally speaking, this algorithm consists of two global steps where each of them is executed in a parallel manner. For the first step we will use the following two positive sets of integers $K = \{k_i\}_{i=2}^{m_p}$ and $N = \{n_i\}_{i=2}^{m_p}$ which are related to each another by

$$n_i = n_{i-1} - k_i$$

and where m_p will be defined later as the number of phases that the first step of this algorithm undergoes till its final termination. Also it is convenient to append to the matrix A by adding to it an $n+1$ column which is equal to the right hand side vector b.

In the begining of the algorithm, as a preparation for the first step, the elements of the matrix A are distributed among the processors in the row-wise direction: PE_0 gets the first $k_1 + k_2$ rows of the matrix, while each of the other processors i gets $\frac{n-k_1-k_2}{p}$ rows of the matrix A, from row number $k_1 + k_2 + \frac{n-k_1}{p}(i-1)+1$ to $k_1 + k_2 + \frac{n-k_1}{p}i$. For convenience, let us define by $SET(s,t)$ the set of equations from equation number s to equation number t. Also let us define the set of rows that PE_i has in the m^{th} phase of the first step of the algorithm as follows:

$$SET_i^m = SET(k_m + k_{m+1} + \frac{n_m}{p}(i-1)+1 \ , \ k_m + k_{m+1} + \frac{n_m}{p}i) \qquad (1)$$

As it was mentioned before, the suggested parallel algorithm consists of two basic steps: the forward step and the backward step, where both are similar to the forward and backward steps of the scalar GE procedure. The parallel algorithm for the forward step is described in table 1 which sometimes called the parallel activity table for this step. Here $r < k_1$ is a positive integer selected by the user, and usually depends also on the characteristics of the parallel machine. Intuitively, if the parallel machine is homogeneous, then we will try to keep r to a minimum ($r = 1$). However, in a hetrogeneous environment, r will probably relate to the relative powers of the processors.

The resultant matrix of this step has the structure of a "staired like" upper triangular matrix. The parallel algorithm for the backward step is based on the parallel algorithm for triangular system devised by *Lin* and *Zhang* [3]. The sequential algorithm for triangular systems may be written as follows:

$$x_1 = \frac{b_1}{a_{11}}; \ x_k = \frac{b_k}{a_{kk}} - \sum_{i=1}^{i=k-1} \frac{a_{ki}}{a_{kk}} x_i; \quad k = 2, 3, \cdots, n. \qquad (2)$$

As in the first step, the idea is to let each PE perform the same amount of computations between two successive communication operations, and to make the redundant computations as small as possible. Those redundant computations results from the fact that the algorithm is parallelized, and some of the operations are needed to be executed not only by one processor (as in the scalar case) but by several of them (whichever needs the

Phase number	Processor 0	Processor $i, 1 \leq i \leq p$
1	1. Executes the GE algorithm for the first r variables. 2. Transmits the results of the r variables to the rest of the processors. 3. Executes the GE algorithm for the next $k_1 - r$ equations.	1. Receives the values of the first r variables. 2. Substitutes the values of the first r variables into its set of SET_1^i equations.
$m \geq 2$	1. Substitutes the values of the last k_{m-1} variables. 2. Executes the GE algorithm for the next k_m equations. 3. Transmits the results of those k_m variables to the rest of the processors.	• Substitutes the values of $k_{m-1} - \delta_{m,2}r$ variables into its SET_i^m equations.
	Transfers to the $i-1$ processor $\frac{k_m}{p}(p-i+1)$ of its first equations, and receives $\frac{k_m}{p}(p-i)$ rows from the $i+1$ processor locating them at the end of its set of rows.	

Table 1: Parallel Activity Table for the Forward Step

appropriate results). This step is executed in $r_p + 1$ phases. Similar to step 1, let us denote by $Q = \{q_i\}_{i=1}^{r_p}$ a sequence of positive integers, such that $\sum_{i=1}^{r_p} q_i < n$. Then a possible parallel backward step is summarized in the parallel activity table 2.

3 The Main Properties of the Algorithm.

One of the more important features of this parallel algorithm is that all the processors are working all the time towards the solution of the problem. More than that. It is possible to minimize the waiting time for needed information or data for each of the processors. This means, that the execution time of all of the processors between two successive communication operations (independent on the task that they ought to perform), will be the same (for all of them). In order to set up the equations for this time-balance requirement, let us define first the following quantities:

- $GE(\alpha, \beta)$ - is the time that it takes for one processor to execute the Gaussian elimination algorithm for α variables with β right hand sides.
- $TR(\alpha, \beta)$ - is the time that it takes to transfer α vectors each of length of β to the neighbor processor.
- $REC(\alpha, \beta)$ - is the time that it takes to receive α vectors each of the length of β by the neighbor processor.
- $MULT(r1, r2, r3)$ - is the time that it takes to multiply two matrices, one of the sixe of $r1 \times r2$ and the other of the size $r2 \times r3$.
- $SUBT(r1, r2)$ - is the time that it takes to subtract two matrices one from the other where their size is $r1 \times r2$.

In terms of the above definitions, if it is desired that the execution time of PE_0 will be equal to the execution time of any of the other PEs in each of the phases of the forward step. This means that for the first phase of this

Phase number	Processor 0	Processor $i, 1 \leq i \leq p$
1	1. Executes the scalar algorithm for the first r variables. 2. Transmits the results of the r variables to the rest of the processors. 3. Executes the scalar algorithm for the next $q_1 - r$ equations.	1. Receives the values of the first r variables. 2. Substitutes the values of the first r variables: $b_j \leftarrow b_j - \sum_{over\ r\ elements} a_{ij}x_j$ where $q_1 + 1 + \lceil (i-2)\frac{n-q_1}{P} \rceil \leq j \leq q_1 + 1 + \lceil (i-1)\frac{n-q_1}{P} \rceil$. 3. Transmits back the new values of b_i's to PE_0.
$m \geq 2$	1. Calculates the values of the next q_m variables, using the latest values of the bs. 2. Transmits the results of those q_m variables to the rest of the processors.	• Substitutes the values of $q_{m-1} - \delta_{m,2}r$ variables into its set of equations. • Reports back to PE_0 the new values for the bs.
$m = r_p + 1$	finishes to solve in a sequential mode the rest of the equations.	

Table 2: Parallel Activity Table for the Backward Step

step we have the following balance:

$$\begin{bmatrix} GE(r,n-r)+TR(r,n-r)+ \\ MULT(k_1-r,r,n-r)+ \\ SUBT(k_1-r,n_1)+GE(k_1-r,n_1) \end{bmatrix} = \begin{bmatrix} REC(r,n-r)+ \\ MULT(\frac{n_2}{p},r,n-r)+ \\ SUBT(\frac{n_2}{p},n-r) \end{bmatrix} \quad (3)$$

Similarly for the m^{th} phase of this step and any processor i, the following time balace holds:

$$\begin{bmatrix} MULT(k_m,k_{m-1},n_{m-1})+ \\ SUBT(k_m,n_{m-1})+ \\ GE(k_m,n_m)+ \\ TR(k_m,n_m)+REC(k_{m+1},n_m) \end{bmatrix} = \begin{bmatrix} MULT(\frac{n_i}{p},k_{m-1}-\delta_{m,2},n_{m-1})+ \\ SUBT(\frac{n_i}{p},n_{m-1})+ \\ TR(\frac{k_m}{p}(p-i+1),n_{m+1})+ \\ REC(\frac{k_m}{p}(p-i),n_{m+1}) \end{bmatrix} \quad (4)$$

Once we are given the appropriate execution time for any of the above operations, those equations determine the sets of integers K and N. These operations depend very much on the basic features of each processor in the parallel engine. In what follows we will assume that the processors are regular and standard scalar processors, while in the extended version of this paper we will present also results for vector processors and parallel processors (means that the engine is of the degenerate parallel system type like a paralleld-vectored machine).

For a scalar processor, let us define the CPU times needed to execute an addition, a multiplication and to find the inverse of a scalar number by A, M and I respectively. Also we will assume the following time complexities:

- if the LU decomposition approach is used than

$$\begin{aligned} GE(r,s) &= M\frac{r}{3}[(s+1)r^2+(3s+1)r-s] \\ &\quad +A\frac{r}{6}[2(s+1)r^2+3(s-1)r+(1-5s)]+Isr. \end{aligned} \quad (5)$$

If the regular elimination is used, than

$$GE(r,s)=(M+A)r(\frac{r^2}{3}+\frac{3rs}{2}+s^2)+Mr(r+2s) \quad (6)$$

- $MULT(r,t,s) = (A+M)rts.$
- $SUBT(r,s) = Ars.$

where the contributions of the broadcasting time complexities will be discussed in the extended version of the paper.

Lets define also by α and β the following quantities:

$$\alpha = A/M \quad ; \quad \beta = I/M \tag{7}$$

It is obvious that $\alpha < 1$ and $\beta > 1$ where for most of the machines β is upper bounded roughly by 10. Then the above balance of the execution time between the master processor, PE_0 and the rest of the processors can be expressed as:

$$a_0 k_m^4 + a_1 k_m^3 + a_2 k_m^2 + a_3 k_m + a_4 = 0 \tag{8}$$

where the LU decomposition version for the GE operation eq.(15) was used. The coefficients of this equation for k_m are given as follows:

$$a_0 = 1+\alpha \tag{9}$$

$$a_1 = -n_{m-1} + 3(1+\frac{\alpha}{2}) \tag{10}$$

$$a_2 = -3(1+\frac{\alpha}{2})n_{m-1} + (1+\frac{5}{2}\alpha + 3\beta) \tag{11}$$

$$a_3 = -n_{m-1}[3\frac{p+1}{p}(k_{m-1}+\alpha) - (1+\frac{5}{2}\alpha - 3\beta)] \tag{12}$$

$$a_4 = 3\frac{p+1}{p}(k_{m-1}+\alpha)n_{m-1}^2 \tag{13}$$

Based on this model for the GE procesure of PE_0 we have the following results for the forward step:

Lemma 1: For large values of n_m, $k_m = \frac{n_{m-1}}{1+\alpha} + O(1)$.

Proof: Assume a solution of the singular pertubation type $k_m = \frac{n_{m-1}}{1+\alpha} + \epsilon$ where ϵ is a weaker function then the leading term in this expression, then it is possible to show that

$$\epsilon = -\frac{3\alpha(1+0.5\alpha) + [(1+2.5\alpha)(2+\alpha) + 3\alpha\beta - 3\frac{p+1}{p}\alpha(1+\alpha)(k_{m-1}+\alpha)](\frac{1+\alpha}{n_{m-1}})}{1+3(1-2\alpha)(1+0.5\alpha)\frac{1+\alpha}{n_{m-1}}}$$

□

In addition, it is possible to find lower bounds for the values of the elements of the sequence K as is given by the following lemma:

Lemma 2: The smallest value of n_{m-1} for which a real solution exists for k_m is approximately $3\alpha(1+\alpha)^4$. For this case, the value of k_m may be approximated by $(1+\alpha)[3\alpha(1+\alpha)k_{m-1}]^{\frac{1}{3}}$.

This lemma, which can be proven easily, leads to the following upper bound on the number of phases that the the forward step requires till its termination.

Lemma 3: The maximum number of phases is upper bounded by $O(C\ \log n)$ where $C \approx \log(1+\alpha)$

Finally, based on the above lemmas, we can prove the following upper bound for the speed-up of the forward step only:

Theorem 1: The maximum speed-up of the forward step cannot exceed $O(\log P)$.

We should realize that this result is for the version of $GE(r,s)$ given by eq. (5) and is modeled in eq. (8). The origin for this poor theoretical speed-up is probably in the bad algorithm for the scalar GE procedure. When using eq.(6) for this procedure, the following recursive equation for the integer set K is obtained:

$$a_0 k_m^3 + a_1 k_m^2 + a_2 k_m + a_3 = 0 \tag{14}$$

where the coefficients a_i are given as follows:

$$a_0 = \frac{2}{3}(1+\alpha) \tag{15}$$

$$a_1 = \frac{3}{2} - (1+\alpha)(n_{m-1} + \frac{2}{3}) \quad (16)$$

$$a_2 = -n_{m-1}[k_{m-1} + 1 + \frac{1}{P}(k_{m-1} + \alpha)] \quad (17)$$

$$a_3 = \frac{1}{P}(k_{m-1} + \alpha)n_{m-1}^2 \quad (18)$$

In the following lemma we try to isolate the proper root for eq.(14).

Lemma 3: Eq.(14) has two positive roots for k_m one which is larger than n_{m-1}, $k_m^{(1)} > n_{m-1}$ and one which is smaller than n_{m-1}, $k_m^{(2)} < n_{m-1}$.

The proof for this lemma is simple, and one of the outcomes of this proof is that asymptotically we can approximate the first root as is given in the following lemma:

Lemma 4: The largest root of eq.(14) is given by

$$k_m^{(1)} = \sqrt{1.5(P+1)\frac{k_{m-1}n_{m-1}}{P}} + \epsilon \quad (19)$$

where ϵ can be approximated by $O(n_{m-1}^{\gamma})$ where $\gamma << 0.5$.

Using this result, it is possible to isolate the second positive root, as it is given in the following lemma:

Lemma 5: $k_{m-1}^{(2)} \approx (1.5\frac{2(P+1)}{3P}k_{m-1}n_{m-1}^3)^{\frac{1}{4}}$.

Although the proof for this lemma involves lengthy algebra, it is quite straight forward. The following table gives some numerical results for the set K for $n = 1000000$, $k_1 = 4$ and $r = 1$, for various number of processors.

P and phase number	i=1	i=2	i=3	i=4	i=5	i=6	i=7
2	1	36042	483121	480835			
10	i=1	i=2	i=3	i=4	i=5	i=6	i=7
	1	24102	294872	421064	259909		
100	i=1	i=2	i=3	i=4	i=5	i=6	i=7
	1	13554	144757	232322	205233	146224	95921
	i=8	i=9	i=10	i=11	i=12	i=13	i=14
	60904	38170	23979	14804	9202	5718	3553
	i=15	i=16	i=17	i=18	i=19	i=20	i=21
	2207	1371	851	528	328	203	126

The first phenomena that can be observed from this table is that the number of phases till termination of the forward step is an increasing function of P. In fact we have

Lemma 6: The number of phases is the forward step is upper bounded by $O(3 \log P)$ plus a very weak function of $\log \frac{n}{k_1}$.

It should be noted that the forward step is terminated for $m = m_p$ where $k_{m_p} \geq n_{m_p}$. From this condition we may have also some intuition to what would be an upper bound for the speed-up of this step. Since the the last phase is a scalar phase, as it is executed only by PE_0, then we have:

Lemma 7: The speed-up of the forward step is upper bounded by $P - (\frac{k_{mf}}{n})^3$.

In the forthcoming paper we will give some more results which are not as important as the above, for example, approximations for the elements of K as function of n, k_1 and P. Those results gain some importance when this method is implemented on an actual parallel machine.

For the backward step, the following equations presents the relations between the successive elements in the sequence Q. For the first phase, the balance of the CPU time is:

$$\frac{q_1(q_1+1)}{2} = \frac{(n-q_1)}{P}. \tag{20}$$

and the balance for the phase m is:

$$q_m q_{m-1} + \frac{q_m(q_m+1)}{2} = \frac{q_{m-1}}{P}(n - \sum_{1}^{m-1} q_i) \tag{21}$$

Also the rows of A matrix that correspond to those $b_i^{(1)}$'s are transferred to the $PE1$. The above equations for the backward step was solved numerically for the case of $P = 2$, and the results are given in the following table:

n	i	1	2	3	4	5	6
10^4	q_i	139	1400	2814	2337	1434	820
	i	7	8	9	10	11	
	q_i	462	260	146	82	46	

n	i	1	2	3	4	5	6
10^3	q_i	43	213	279	197	116	66
	i	7	8	9			
	q_i	37	21	12			

In the following section we will prove that the running time of this procedure for determining the integer sequence Q is also small since the number of phases of this step till termination is: $r_P = O[\ Plog(n)\ ]$. To obtain the order of of the number of phases r_P, let's study a slightly different recursive system for u, which is defined by the following equations:

$$\frac{u_1^2}{2} = \frac{n-u_1}{P-1} \tag{22}$$

$$u_{i-1}u_i + \frac{u_i^2}{2} = \frac{u_{i-1}(n - \sum_{j=1}^{i} u_i)}{P-1},\ (for\ i \geq 2) \tag{23}$$

Define: $N_i = \frac{n-\sum_{j=1}^{i-1} u_j}{P-1}$, then we have $N_{i+1} = u_{i+1}(1 + \frac{1}{P-1} + \frac{u_{i+1}^2}{2U_i})$, and

using $N_{i+1} = N_i - \frac{u_i}{P-1}$, we get finally $u_{i+1}(1 + \frac{1}{P-1}) + \frac{u_{i+1}^2}{2u_i} = u_i + \frac{u_i^2}{2u_{i-1}}$. Define: $t_i = \frac{u_{i+1}}{u_i}$, then the following equation which is equivalent to eq. (14) is obtained:

$$\frac{t_i^2}{2} + t_i(1 + \frac{1}{P-1}) - (1 + \frac{t_{i-1}}{2}) = 0 \qquad (24)$$

Solving the equation (24) for t_i, we get $t_i = \sqrt{(1 + \frac{1}{P-1})^2 + 2 + t_{i-1}} - (1 + \frac{1}{P-1})$. Define: $\beta_i = t_i + 1 + \frac{1}{P-1}$, we have $\beta_i = \sqrt{(\frac{1}{P-1})(1 + \frac{1}{P-1}) + 2 + \beta_{i-1}}$. Now the following lemma is obvious.

Lemma 7: $L_\beta = \lim_{n\to\infty} \beta_i = \frac{1}{2} + \sqrt{\frac{9}{4} + \frac{1}{P-1}(1 + \frac{1}{P-1})}$

Now let's define $\mu = \sqrt{1 + (6 - \frac{2}{P-1})\frac{1}{L_\beta}}$, $\sqrt{[\frac{1}{P-1}(1 + \frac{1}{P-1}) + 2]\frac{1}{L_\beta} + 1}$, and $R = log(L_\beta + \epsilon)$. The following lemma gives an estimate of the speed of the convergence of the sequence $\{\beta_i\}$.

Lemma 8: Given $\epsilon > 0$, let $N_\epsilon = log(log(\beta_2)) - log(R)$, then $\beta_i < L_\beta + \epsilon$ for $i > N_\epsilon$.

Proof: Simply by realizing first that $log(L_\beta + \epsilon) - 2log(\mu) > 0$, and secondly, since $\beta_i \geq L_\beta$, then $\beta_i \leq \mu\sqrt{\beta_{i-1}}$.

□

From the definition of β_i, we easily see now that $t_i < L_\beta - (1 + \frac{1}{P-1}) + \epsilon$ for $i > N_\epsilon$.

Lemma 9: $u_i \leq u_1$ when $i > O(\, Plog(n)\,)$.

Proof: We choose ϵ such that $L_\beta - (1 + \frac{1}{P-1}) + \epsilon < 1 - \gamma$, where γ is a fixed number, and $0 < \gamma < 1$. Consider the equation $L_\beta - (1 + \frac{1}{P-1} + \epsilon^* = 1$, then $\epsilon^* = 1.5 + \frac{1}{P-1} - \sqrt{\frac{9}{4} + \frac{1}{P-1}(1 + \frac{1}{P-1})} = O(\frac{1}{(P-1)})$. If we let $\gamma = \frac{\epsilon^*}{2}$ and $\epsilon = \gamma$, then we have $t_i \leq 1 - \gamma$ when $i > N_\epsilon$. That is $u_{i+1} < (1 - \gamma)u_i$ when $i > N_\epsilon$. From $log(R) = O(\, log(P)\,)$ and $u^{i+k} \leq (1 - \gamma)^k u_i$ for $i > N_\epsilon$, we can easily see the lemma is true.

□

Theorem 2: $r_p < O(P \log n)$.

Proof: By induction it can be shown that $\frac{q_i}{u_i} \leq 1 - n^{-\frac{3}{4}}$. Using this result, a lemma for k_i which is similar to lemma 3 for u_i can be formulated, and the result follows.

□

4 Analysis of The Algorithm

Here we give a simple analysis of the parallel algorithm subject to certain models for the communication complexity, see [5] for more information. In the final version of the paper, we will consider a broader range of models for communication. Let us define the time needed to broadcast a packet of information of size N as $t_N = \alpha + \Gamma_P N$, where α is start up time, and Γ_P is a function of P. $\Gamma_P = 1$ if the information is sent to the immediate neighbors, and $\Gamma_P = P$ if the information is sent to all other P PEs. We define $T_c(n)$ as the communication time of the parallel algorithm.
In the parallel algorithm, PE_0 has to send its result to all other P processors at each phase of the forward step and all the other processors send only the result in the worst case to the immediate neighbors. We can easily get an upper bound for $T_c(n)$.

Theorem 3: for the forward and the backward steps $T_c(n) \leq O(\ Pn + \frac{n^2}{2(P-1)}\ )$.

Let's define by $T_1(n)$ the computational sequential time and by $T_P(n)$ the computational parallel time using P PEs. Let $S_P(n) = \frac{T_1(n)}{T_P(n)}$ denote the speed up and $\eta_P = \frac{S_P(n)}{P}$ denote efficiency.
Definition: It is said that the speed up is optimal when $\lim_{n\to\infty} S_P(n) = P$, and the efficiency is optimal when $\lim_{n\to\infty} \eta_P(n) = 1$.

Theorem 3: The present parallel algorithm has optimal speed up and optimal efficiency.
Proof: From the result of theorem 2 and $T_1(n) = \frac{n^2}{2}$, we can easily reduce to the results.

□

5 References

[1] *Dongarra, J. J., Gustavson, F. G.* and *Karp, A.*, (1984), "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline

Machine", *SIAM Rev.*, 26, pp.91-112.
[2] *George, A., Heath, M. T.* and *Liu, J.*, (1986), "Parallel Cholesky Factorization on a Shared Memory Multiprocessor", *Linear Algebra and its Applications*, 77, pp. 165-187.
[3] *Zhang, H.* and *Lin, A.*, (1988), "A Parallel Algorithm for *LU* Decomposition", to appear in the Proceeding of *the Second International Conference on Vector and Parallel Computing*", Bergen, 1988.
[4] *Lin, A.* and *Zhang, H.*, (1987), "A New Parallel Algorithm for Linear Triangular Systems", to appear in the Proceeding of *the Third SIAM Conference on Parallel Processing for Scientific Computing*, Dec. 1987.
[5] *Saad, Y*, (1986), "Communication Complexity of the Gaussian Elimination Algorithm on Multiprocesors", *Linear Algebra and its Applications*,77 , pp.315-340.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-103229 ICOMP-90-18		

4. Title and Subtitle	5. Report Date
Parallel/Distributed Direct Method for Solving Linear Systems	July 1990
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Avi Lin	E-5645
	10. Work Unit No.
	505-62-21

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191	
	13. Type of Report and Period Covered
	Technical Memorandum

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546-0001	

15. Supplementary Notes

Avi Lin, Department of Mathematics, Temple University, Philadelphia, Pennsylvania 19122 and Institute for Computational Mechanics in Propulsion, Lewis Research Center (work funded by NASA Space Act Agreement C-99066-G). Space Act Monitor: Louis A. Povinelli.

16. Abstract

In this paper, a new family of parallel schemes for directly solving linear systems will be presented and analyzed. It is shown that these schemes exhibit a near optimal performance and enjoy several important features:

1. For large enough linear systems, the design of the appropriate parallel algorithm is insensitive to the number of processors as its performance grows monotonically with them.
2. It is especially good for large matrices, with dimensions large relative to the number of processors in the system. In this case, it achieves
   - an optimal speed up.
   - an optimal efficiency.
   - a very low communication time complexity.
3. It can be used in both distributed parallel computing environments and tightly coupled parallel computing systems.
4. This set of algorithms can be mapped onto any parallel architecture without any major programming difficulties or algorithmical changes.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parallel computations Numerical solution of linear systems	Unclassified – Unlimited Subject Category 64

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	18	A03

*For sale by the National Technical Information Service, Springfield, Virginia 22161