Pergamon

Computers Math. Applic. Vol. 29, No. 7, pp. 39–54, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00017-8

A New Gaussian Elimination-Based Algorithm for Parallel Solution of Linear Equations

K. N. BALASUBRAMANYA MURTHY
AND
C. SIVA RAM MURTHY*
Department of Computer Science and Engineering
Indian Institute of Technology, Madras-600 036, India
*murthy@iitm.ernet.in

(Received February 1994; accepted July 1994)

Abstract—In this paper, a variant of Gaussian Elimination (GE) called Successive Gaussian Elimination (SGE) algorithm for parallel solution of linear equations is presented. Unlike the conventional GE algorithm, the SGE algorithm does not have a separate back substitution phase, which requires $O(N)$ steps using $O(N)$ processors or $O\left(\log_2^2 N\right)$ steps using $O\left(N^3\right)$ processors, for solving a system of linear algebraic equations. It replaces the back substitution phase by only one step division and possesses numerical stability through partial pivoting. Further, in this paper, the SGE algorithm is shown to produce the diagonal form in the same amount of parallel time required for producing triangular form using the conventional parallel GE algorithm. Finally, the effectiveness of the SGE algorithm is demonstrated by studying its performance on a hypercube multiprocessor system.

Keywords—Linear equations, Triangulation, Back substitution, Gaussian elimination, Numerical stability, Pivoting, Task system, Scheduling, Multiprocessor system.

1. INTRODUCTION

The problem of solving a set of linear algebraic equations $Ax = b$ (where A is a known $N \times N$ matrix, x and b are unknown and known N vectors, respectively) is one of the central problems in computational mathematics and computer science. Efficient numerical methods for solving this problem on uniprocessor systems have been developed, and many reliable and high quality codes are available for different cases of linear systems. Recent advances in VLSI and networking technology have led to widespread interest in the use of multiprocessor systems for solving many practical problems. Bertsekas and Tsitsiklis [1], Heller [2], Lakshmivarahan and Dhall [3], and Sameh and Kuck [4], describe the current state of art in parallel numerical algebra. In this paper, we present a new algorithm called Successive Gaussian Elimination (SGE) for solving dense system of linear algebraic equations on multiprocessor systems. Most importantly, the algorithm permits partial pivoting to improve numerical stability. The SGE algorithm is essentially a variant of the Gaussian Elimination (GE) algorithm and does not require a separate back substitution phase to find the complete solution vector. It may be noted that the back substitution phase requires $O(N)$ steps using $O(N)$ processors or $O(\log_2^2 N)$ steps using $O(N^3)$ processors [2,4].

The rest of the paper is organized as follows. In the next section, we first define the problem and then discuss the relevant work. Section 3 presents the SGE algorithm for the problem. In Section 4, the memory requirements and error analysis of the SGE algorithm are described.

This work was supported by the Indian National Science Academy and the Department of Science and Technology.

Section 5 describes, in detail, a method for scheduling the computational tasks in the algorithm onto the processors for efficient implementation on a multiprocessor system. Section 6 presents the performance evaluation of the SGE algorithm. Finally, in Section 7, we present our conclusions.

2. PROBLEM DEFINITION AND RELATED WORK

The problem in solving a set of linear algebraic equations is to find the vector x in the equation $Ax = b$, where A is an $N \times N$ matrix, x is an unknown N vector, and b is a known N vector.

The solution of $Ax = b$ can be obtained by using classical methods such as GE, Gauss-Jordan (GJ), Cramer's rule (CR), and LU decomposition [1–12]. The solution process of $Ax = b$ by these methods (except GJ and CR algorithms) essentially consists of triangulation phase followed by back substitution phase. Therefore, the total time taken for solving the problem on a multiprocessor system is the sum of parallel times taken for triangulation and back substitution phases. Further, the classical methods require pivoting to assure numerical stability.

Most of the existing algorithms available in the literature for parallel solution of linear equations consider only the computational intensive triangulation phase assuming that efficient algorithms exist for the simple back substitution phase. However, it is important to note that both the phases using different efficient algorithms may not optimally be implemented on any given multiprocessor system as these algorithms were developed for different multiprocessor configurations.

Recently, two back substitution free algorithms based on GJ and CR are discussed in [10,12]. The parallel CR algorithm [12], which does not support pivoting, is applicable only for diagonally dominant systems while the parallel GJ algorithm [10] permits the partial pivoting in which the maximum element is found among the subdiagonal elements of the pivot column and the pivot column element (instead of finding the maximum element in the entire pivot column).

In this paper, we present a back substitution-free SGE algorithm, which supports partial pivoting, to produce the diagonal form in $O(N^2)$ steps using $O(N)$ processors against the same number of steps required for producing the triangular form in the existing methods.

3. SUCCESSIVE GAUSSIAN ELIMINATION (SGE) ALGORITHM

It is clear while solving $Ax = b$ that the value of x_i depends on the value of x_k $(k = 1, 2, \ldots, N$ and $k \neq i)$ indicating $(N-1)^{\text{th}}$ level dependency. It is obvious, in the GE method, the value of x_i $(i = 1, 2, \ldots, N)$ is found by eliminating its dependency on x_k for $(k < i)$ in the triangulation phase and x_k for $(k > i)$ in the back substitution phase.

In the SGE algorithm, the dependencies of all the unknowns are reduced to half at every stage and finally to zero in $\log_2 N$ stages (i.e., N linear independent equations at Stage 1 are replaced by two sets of $N/2$ linear independent equations at Stage 2, by four sets of $N/4$ linear independent equations at Stage 3, etc.) which is accomplished by using the concept of forward (left to right) and backward (right to left) eliminations.

In this section, we explain how to obtain the diagonal form of coefficient matrix A in the equation $Ax = b$. For better exposition, we assume that $N = 2^\alpha$, where α is an integer and later, we relax this assumption. The proposed algorithm consists of the following steps.

STEP 1. We form two matrices namely A_0 and A_1 identical to the coefficient matrix A and find the maximum element in the pivot column of A_0 and A_1, and exchange the pivot row with the row in which maximum element is found.

STEP 2. Using the GE method (subtracting fractions of pivot row elements from nonpivot elements), we triangulate A_0 in the forward direction to eliminate the subdiagonal elements in the pivot columns (note that partial pivoting is carried out before each column is being eliminated) $1, 2, 3, \ldots, N/2$ (by taking $a_{11}, a_{22}, \ldots, a_{N/2,N/2}$ as pivot elements) to reduce its order to $N/2$ (ignoring the eliminated columns and the corresponding rows). Concurrently, we triangulate A_1 in the backward direction to eliminate the superdiagonal elements in the pivot columns (again

note that partial pivoting is carried out before any column is being eliminated) N, $N-1$, $N-2, \ldots, N/2+1$ (by taking a_{NN}, $a_{N-1,N-1}, \ldots, a_{N/2+1,N/2+1}$ as pivot elements) to reduce the order of A_1 also to $N/2$ (again ignoring the eliminated columns and the corresponding rows). With this, modified A_0 may be treated as a new coefficient matrix (we call it reduced A_0) with columns and rows $N/2+1$, $N/2+2, \ldots, N$ and, similarly, modified A_1 as a new coefficient matrix (we call it reduced A_1) with columns and rows $1, 2, \ldots, N/2$.

STEP 3. We duplicate the reduced matrices A_0 to form A_{00} and A_{01} , and A_1 to form A_{10} and A_{11} (each of these duplicated matrices will be of the same order, i.e., $N/2$). We now (note that partial pivoting is carried out before each column being eliminated) triangulate A_{00} and A_{10} in the forward direction, and A_{01} and A_{11} in the backward direction through $N/4$ pivot columns using GE, thus reducing the order of each of these matrices to half of their original size, i.e., $N/4$. Note that the above four matrices are reduced in parallel.

STEP 4. We continue this process of halving the size of submatrices (using partial pivoting at each step) and doubling the number of submatrices for $\log_2 N$ times so that we end up with N submatrices each of order 1. These N submatrices represent the diagonal form of the coefficient matrix A in the equation $Ax = b$. To obtain an idea about the nature of computations during each step of the SGE algorithm, we give procedures for forward elimination and backward elimination below by taking the order of the matrix as m.

```
PROCEDURE forward-elimination;
BEGIN
FOR k = 1 TO m/2 DO
[ BEGIN                                                          ]
    find q such that                                             ]
    |a_{q,k}| = max of (|a_{k,k}|, |a_{k+1,k}|, ..., |a_{m,k}|)  ] T_k^k
    exchange row k and row q                                     ]
    FOR q = (k+1) TO m DO                                        ]
      a_{q,k} = a_{q,k}/a_{k,k}                                  ]
    FOR j = (k+1) TO m DO                                        ]
      FOR i = (k+1) TO m DO                                      ] T_k^j
        a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}                    ]
[ END (* of for loop with index k *)
END(* of procedure forward-elimination *)
PROCEDURE backward-elimination;
BEGIN
  FOR k = m DOWNTO (m/2+1) DO
[ BEGIN                                                          ]
    find q such that                                             ]
    |a_{q,k}| = max of (|a_{k,k}|, |a_{k-1,k}|, ..., |a_{1,k}|)  ] T_{m-k+1}^{*k}
    exchange row k and row q                                     ]
    FOR q = (k-1) DOWNTO 1 DO                                    ]
      a_{q,k} = a_{q,k}/a_{k,k}                                  ]
    FOR j = (k-1) DOWNTO 1 DO                                    ]
      FOR i = (k-1) DOWNTO 1 DO                                  ] T_{m-k+1}^{*j}
        a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}                    ]
[ END (* of for loop with index k *)
END(* of procedure backward-elimination *)
```

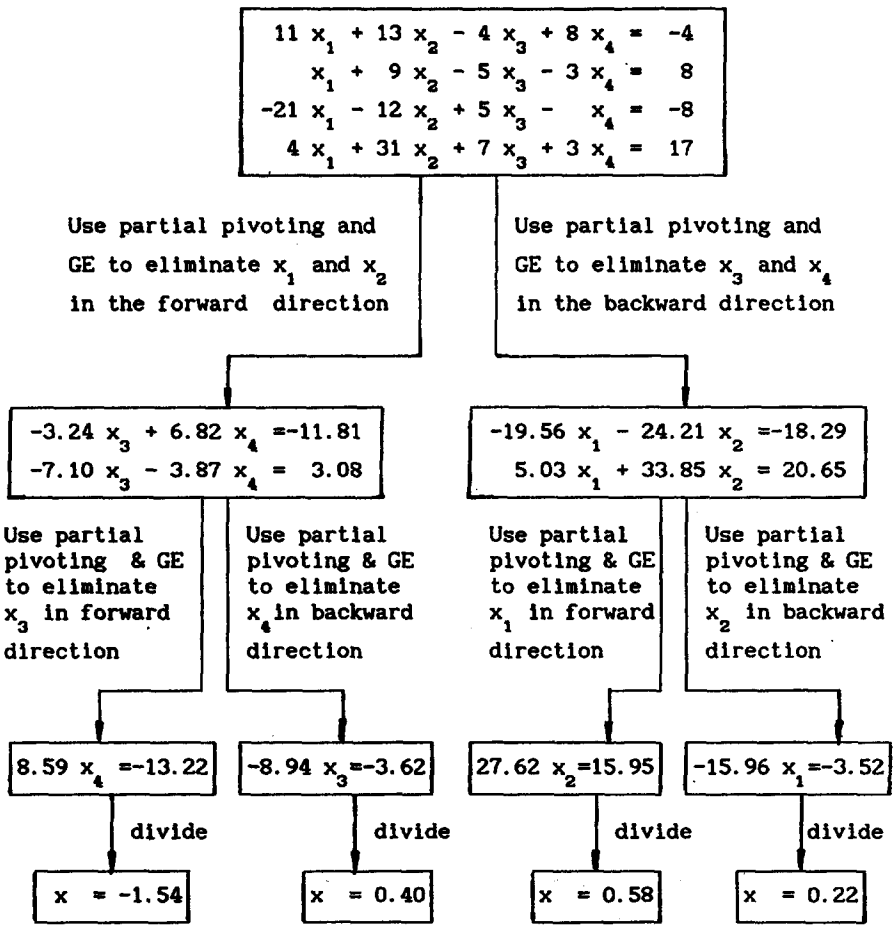

Figure 1. Solving a 4 × 4 system using the SGE algorithm.

The idea of forward elimination (left to right) and backward elimination (right to left) is demonstrated in Figure 1 for solving a 4 × 4 system. For $N = 8$, we have shown the algorithm progression for producing the diagonal form in the form of a binary tree in Figure 2. Figure 3 shows the algorithm progression for producing the diagonal form for $N = 11$, where $N \neq 2^{\alpha}$ (α is an integer).

4. MEMORY REQUIREMENTS AND ERROR ANALYSIS OF THE SGE ALGORITHM

In this section, we discuss the memory space required by the SGE algorithm and its error analysis in comparison with the GE algorithm.

4.1. Memory Requirements

We now show that the SGE algorithm requires only twice the memory required by the GE algorithm in the actual implementation. Initially, we start with two $N \times (N+1)$ matrices which we refer to as forward matrix (in which b vector is appended as $(N+1)^{\text{th}}$ column) and backward matrix (in which b vector is appended as 0^{th} column) for storing the matrix A and the b vector. The upper triangular part of the different submatrices housed in the forward matrix and the lower triangular part of different submatrices housed in the backward matrix become free as the algorithm progresses and may be used for storing the duplicated versions of the reduced matrices/submatrices (duplication of reduced matrices is essential for avoiding the back

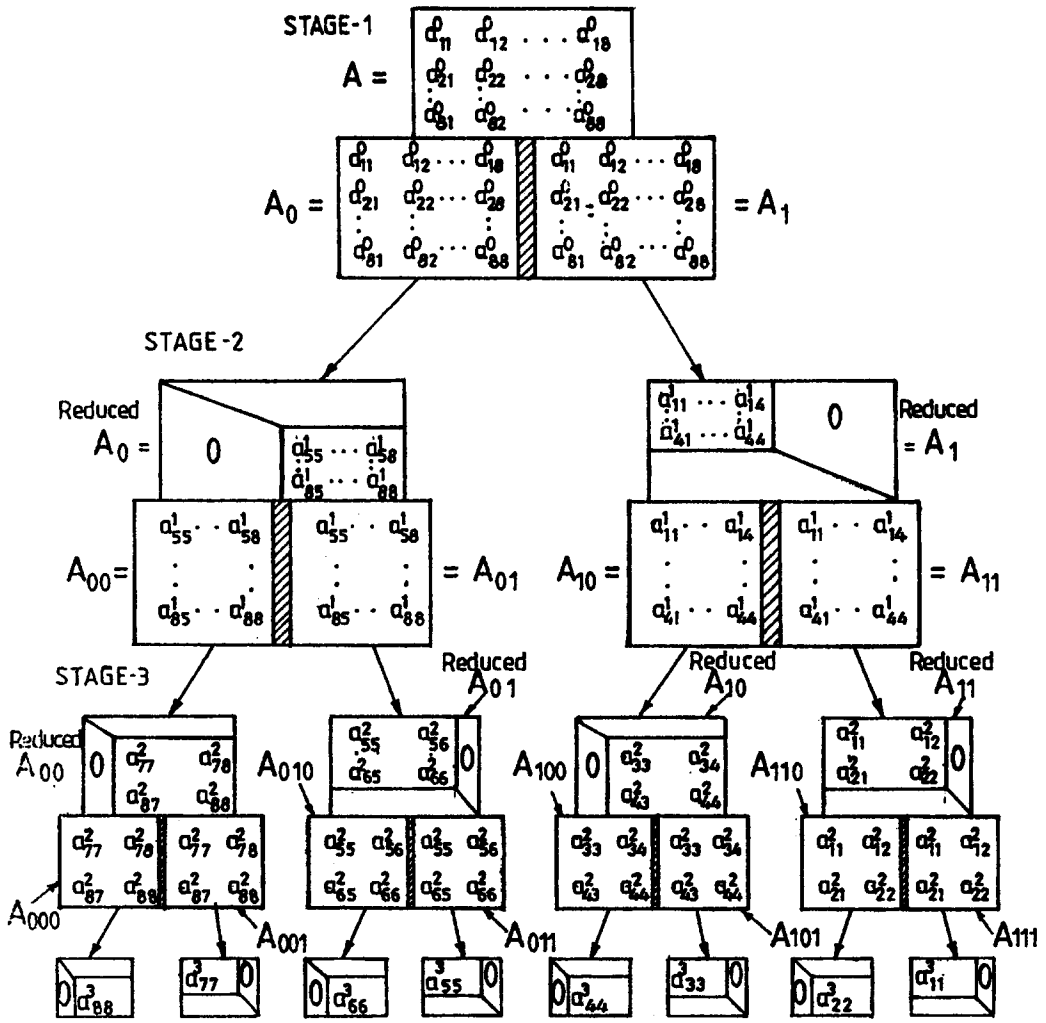

Figure 2. Algorithm progression for $N = 8$ to produce the diagonal form.

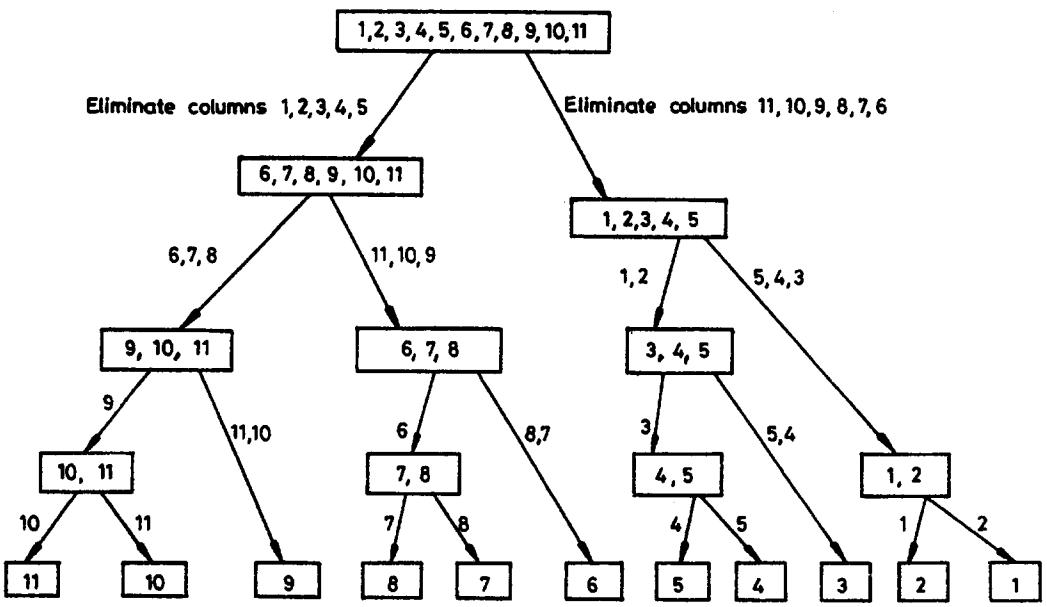

Figure 3. Algorithm progression for $N = 11$.

substitution phase in the SGE algorithm) generated during the progression of the algorithm. At the end of $\log_2 N$ stages, either all the submatrices (each of size 1×2) in the forward matrix are copied into the backward matrix or vice versa for the division step to be carried out in parallel to produce the complete solution vector. For the sake of convenience, it is assumed that the desired columns in the submatrices housed in the forward matrix and the desired columns in the submatrices housed in the backward matrix are eliminated in the forward and backward directions, respectively. Further, we wish to point out that the duplication/copying of submatrices does not require extra time in the actual implementation. With this effective reuse of matrix positions, it is possible to restrict the memory requirements of the new algorithm to twice the memory required by the GE algorithm. In Figures 4 and 5, we demonstrate the reuse of memory allocated for the forward matrix and the backward matrix for a system of four linear equations (The above described SGE algorithm consists of only two stages for a system of four equations).

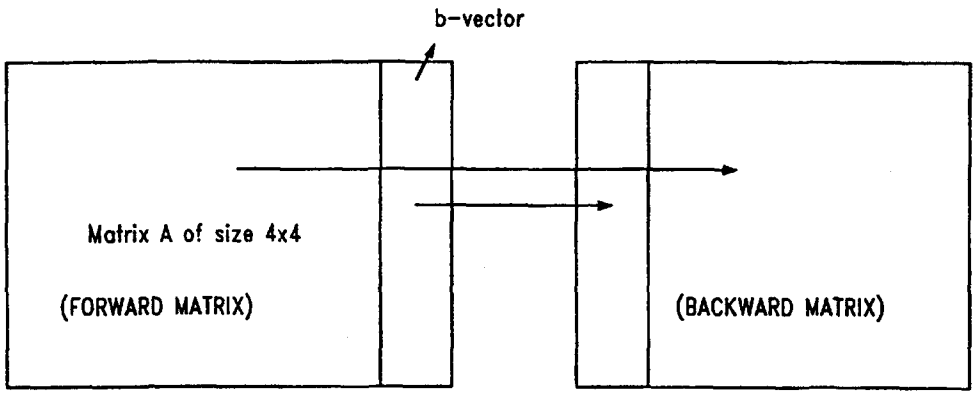

Figure 4. Copying of matrices at the beginning of Stage-1 for $N = 4$.

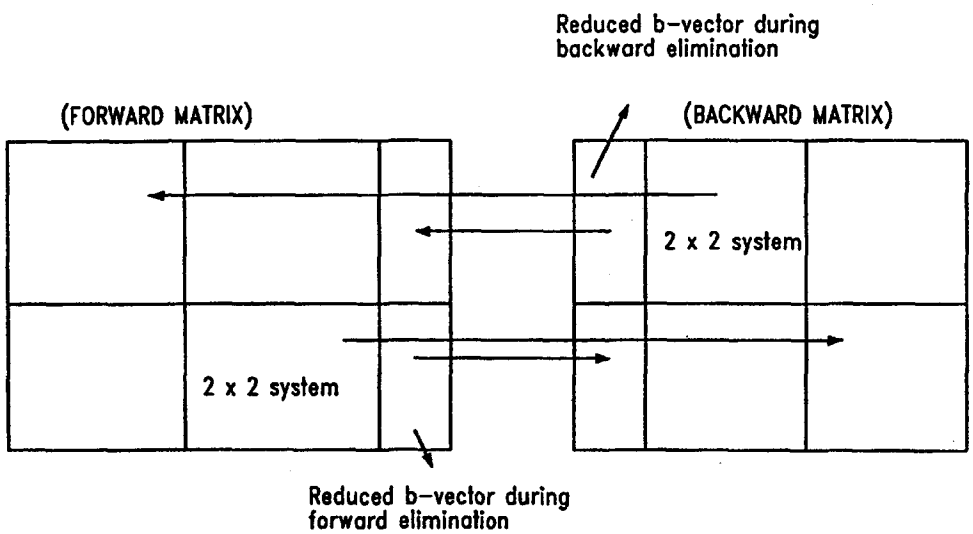

Figure 5. Copying of submatrices at the beginning of Stage-2 for $N = 4$.

4.2. Error Analysis

There are three different methods of error analysis namely, forward error analysis, backward error analysis, and experimental error analysis. We use the elegant backward error analysis (in which computed solution is checked for its closeness to the solution of the original problem) for computing the error bounds in both the GE and SGE algorithms. In the following discussion, we assume that the computations are carried out on a machine with t bits precision for representing the mantissa part of a floating point number.

We now give the expressions for the error bounds in the GE and SGE algorithms, and show that the SGE algorithm has less round-off errors than that of the GE algorithm by using standard floating point computation model for error estimation, namely

$$\text{fl } (a \text{ op } b) = (a \text{ op } b)(1+\epsilon),$$

where fl $(a$ op $b)$ represents the computer version of a floating point operation,

a and b are floating point numbers,
op denotes the actual floating point operation, and
ϵ represents the error in computation.

$\epsilon \leq 0.5\,\beta^{(1-t)}$ for rounded arithmetic (β is the base).

Consider a system of linear algebraic equations $Ax = b$, and let x_t be the true solution and x_c be the computed solution. Let E be the error matrix due to finite digit computed solution. If the computations are exact, then

$$Ax_t = b \quad \text{or} \quad x_t = A^{-1}b,$$

otherwise, $(A+E)x_c = b$ or $x_c = (A+E)^{-1}b$.

Assuming that $(A+E)$ is nonsingular and $\|(A+E)^{-1}\| < 1$, then the relative error in solving $Ax = b$ as per error estimation theorem [7] is given by

$$\frac{\|x_c - x_t\|}{\|x_t\|} = \frac{\left\|(A+E)^{-1}b - A^{-1}\,b\right\|}{\|A^{-1}b\|} \leq \frac{\left\|A^{-1}E\right\|}{1 - \|A^{-1}E\|} \quad \text{or} \quad \frac{\|E\|K(A)/\|A\|}{\{1 - (\|E\|\,K(A)/\|A\|)\}},$$

where $K(A)$ is the standard condition number given by $\left\|A^{-1}\right\|\|A\|$.

It is well-known that the GE algorithm consists of the triangulation phase followed by the back substitution phase. Accordingly, the error matrix E is the sum of two matrices, namely the triangulation error matrix (F) and the back substitution error matrix (B), i.e., $E = F + B$. The relative error in the GE algorithm is therefore given by

$$\frac{\|x_c - x_t\|}{\|x_t\|} \leq \frac{\left\|A^{-1}(F+B)\right\|}{1 - \|A^{-1}(F+B)\|} \quad \text{or} \quad \frac{\|F+B\|K(A)/\|A\|}{\{1 - (\|F+B\|\,K(A)/\|A\|)\}}.$$

In the SGE algorithm, we have only the triangulation phase, therefore, the error matrix consists of only the triangulation error matrix (F), i.e., $E = F$. The relative error in the SGE algorithm is therefore given by

$$\frac{\|x_c - x_t\|}{\|x_t\|} \leq \frac{\left\|A^{-1}\,F\right\|}{1 - \|A^{-1}\,F\|} \quad \text{or} \quad \frac{\|F\|\,K(A)/\|A\|}{\{1 - (\|F\|\,K(A)/\|A\|)\}}.$$

As the round-off errors enter a computation in an additive manner and $F \leq (F+B)$, we can easily conclude that the relative error in the SGE algorithm is less than or equal to the relative error in the GE algorithm.

5. SCHEDULING OF COMPUTATIONAL TASKS IN THE SGE ALGORITHM

One of the key issues in multiprocessing is the distribution of parallel tasks among the various processors in a multiprocessor system to minimize the execution time of the tasks. In this section, we provide an efficient scheduling scheme for assigning the computational tasks in the SGE algorithm to the processors in a multiprocessor system assuming negligible inter-task communications.

From the binary tree representation of the solution process (as shown in Figure 2 for $N = 8$), we make the following observations.

(a) There are $\alpha(= \log_2 N)$ stages with 2^{s-1} submatrices at any stage s each of order $2^{\alpha-s+1}$. Each submatrix is duplicated and reduced to half its size through forward and backward eliminations. From hereafter, we call these submatrices as matrix nodes.

(b) There are α matrix nodes along any path from the root matrix node to a leaf matrix node (including the root matrix node at Stage 1 and excluding the leaf matrix node at Stage $\alpha + 1$).

As there are no standard task sizes (since task size varies from a program segment to an arithmetic operation and is mostly decided on the structure of the multiprocessor used for executing the task system without violating the precedence constraints), in our model, we assume a task to represent either a set of comparisons and divisions in the pivot column or a set of update operations (subtraction and multiplication) in a nonpivot column (as marked in the forward and backward elimination procedures). The computational tasks, along with their precedence constraints in the algorithm, may be represented as a task system. Further, we assume that each of the basic operations namely, comparison, division, multiplication, addition, and subtraction takes one unit of time. Instead of considering the task system of the entire algorithm, we examine the task system of only one matrix node and its scheduling onto processors. This is because the task system and scheduling of tasks among the processors are essentially of the same nature for all the matrix nodes.

Let m be the order of a matrix node which is duplicated and reduced to the order $m/2$ by eliminating $m/2$ pivot columns using GE in the forward direction on one copy and in the backward direction on the other copy. We denote the task system of forward elimination as $(J_m, <)$ and backward elimination as $(J_m^*, <^*)$, where J_m and J_m^* represent the set of tasks in forward and backward eliminations, respectively.

$<$ and $<^*$ indicate the precedence relations among the computational tasks in forward and backward eliminations, respectively.

The above-mentioned task systems are described by the following expressions.

Forward elimination task system is $(J, <)$.

$$\begin{aligned} J_m &= \left\{ T_k^j \mid 1 \le k \le \frac{m}{2} \quad \text{and} \quad k \le j \le m \right\} \\ <&= \left\{ \left(T_k^k, T_k^j \right) \mid 1 \le k \le \frac{m}{2} \quad \text{and} \quad k+1 \le j \le m \right\} \\ &\quad U \left\{ \left(T_k^j, T_{k+1}^j \right) \mid 1 \le k \le \left(\frac{m}{2} - 1 \right) \quad \text{and} \quad k+1 \le j \le m \right\}. \end{aligned}$$

Backward elimination task system is $(J^*, <^*)$.

$$\begin{aligned} J_m^* &= \left\{ T_{m-k+1}^{*j} \mid m \ge k \ge \frac{m}{2} + 1 \quad \text{and} \quad k \ge j \ge 1 \right\} \\ <^*&= \left\{ \left(T_{m-k+1}^{*k}, T_{m-k+1}^{*j} \right) \mid m \ge k \ge \frac{m}{2} + 1 \quad \text{and} \quad k - 1 \ge j \ge 1 \right\} \\ &\quad U \left\{ \left(T_{m-k+1}^{*j}, T_{m-k+2}^{*j} \right) \mid m \ge k \ge \left(\frac{m}{2} + 2 \right) \quad \text{and} \quad k - 1 \ge j \ge 1 \right\}. \end{aligned}$$

We define the following task sequences for the purpose of scheduling.

For forward elimination:

$$\begin{aligned} t_1 &= \left\{ T_1^1, T_1^2, T_2^2, T_2^3, T_3^3, \ldots, T_{m/2-1}^{m/2}, T_{m/2}^{m/2}, T_{m/2}^{m/2+1} \right\} \\ t_j &= \left\{ T_1^j, T_2^j, \ldots, T_q^j \mid q = \min\left(j - 2, \frac{m}{2} \right) \right\} \qquad j = 3, 4, \ldots, m. \end{aligned}$$

For backward elimination:

$$t_m^* = \left\{T_1^{*m}, T_1^{*m-1}, T_2^{*m-1}, \ldots, T_{m/2-1}^{*m/2+1}, T_{m/2}^{*m/2+1}, T_{m/2}^{*m/2}\right\}$$

$$t_j^* = \left\{T_1^{*j}, T_2^{*j}, \ldots, T_q^{*j} \mid q = \min\left(\frac{m}{2}, m-j-1\right)\right\}, \qquad j = m-2, m-3, \ldots 1.$$

Let us assume that m processors are available, i.e., $p_1, p_2, \ldots, p_m$, and we allocate $m/2$ processors, (i.e., $p_{m/2+1}, p_{m/2+2}, \ldots, p_m$) to forward elimination and the remaining $m/2$ processors (i.e., $p_1, p_2, \ldots, p_{m/2}$) to backward elimination task systems. The allocation of task sequences to the processors is as follows.

When m is even

$p_{m/2+1}$ executes t_1

p_k executes $t_{k-m/2+1}$ and t_k, $\quad k = m, m-1, \ldots, m/2+2,$

$p_{m/2}$ executes t_m^*

p_k executes $t_{k+m/2-1}^*$ and t_k^*, $\quad k = 1, 2, 3, \ldots m/2-1;$

When m is odd:

$p_{m/2+1}$ executes t_1

p_k executes $t_{k-m/2}$ and t_k, $\quad k = m, m-1, \ldots, m/2+3,$

$p_{m/2}$ executes t_m^*

p_k executes $t_{k+m/2-1}^*$ and t_k^*, $\quad k = 1, 2, 3, \ldots m/2-1;$

$p_{m/2+2}$ executes $t_{m/2+2}$ and t_{m-2}^*.

The value of m is $N, N/2, N/4, \ldots, 2$ at Stages $1, 2, 3, \ldots, \log_2 N$, respectively. The scheduling of all the tasks in the algorithm onto N processors in the given multiprocessor system is obtained as follows.

At Stage 1, we have one matrix node of order N and, therefore, $N/2$ processors are allocated to forward elimination tasks and the remaining $N/2$ to backward elimination tasks. At Stage 2, we have two matrix nodes of order $N/2$ each and $N/4$ processors are allocated to each of the forward elimination task systems of the two matrix nodes and another $N/4$ processors to each of the backward elimination task systems of the two matrix nodes. For the remaining stages, the same strategy is extended.

For $N = 8$, the task system and its schedule on eight processors are shown in Figures 6 and 7, respectively. The time taken on a uniprocessor system for producing the diagonal form using GJ algorithm (with each basic operation taking one unit of time) is known to be $2N^3 + O(N^2)$.

The time taken on a multiprocessor system with N processors, for producing the diagonal form of the coefficient matrix A of order N by using the SGE algorithm, can be obtained by counting the number of basic operations/time units along the longest path in the task graph of the algorithm. The longest path in the task graph is $\{T_1^1, T_1^2, T_2^2, \ldots, T_{N-1}^{N-1}, T_{N-1}^N\}$ (see Figure 6 for $N = 8$).

Time taken on a multiprocessor system with N processors for producing the diagonal form of a matrix of order N is given by

$$\sum_{K=1}^{N-1} 2(N-k)\,(\text{for pivot column}) + 2(N-k)\ (\text{for nonpivot column}) = 2N^2 + O(N).$$

At this point, it is worth noting that the time taken on a multiprocessor for producing the triangular form using the conventional GE algorithm is also equal to $2N^2 + O(N)$.

Thus, the speedup of the SGE algorithm is defined as the ratio of the time taken on a uniprocessor to produce diagonal form using GJ algorithm and the time taken on a multiprocessor to produce diagonal form using SGE algorithm and is given by

$$\frac{N^3 + O(N^2)}{2N^2 + O(N)} \cong \frac{N}{2}.$$

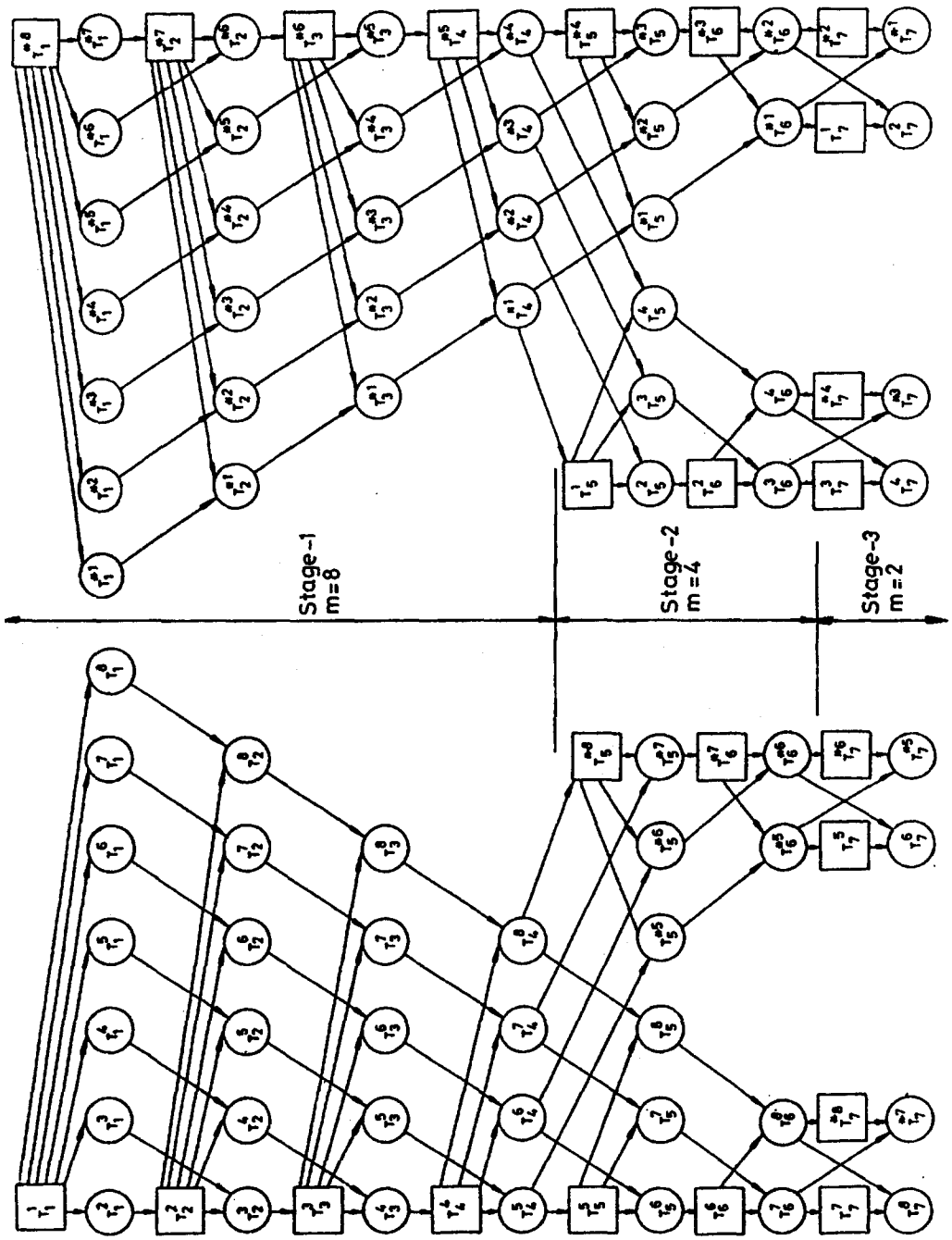

Figure 6. Task system of the SGE algorithm for $N = 8$ to produce the diagonal form.

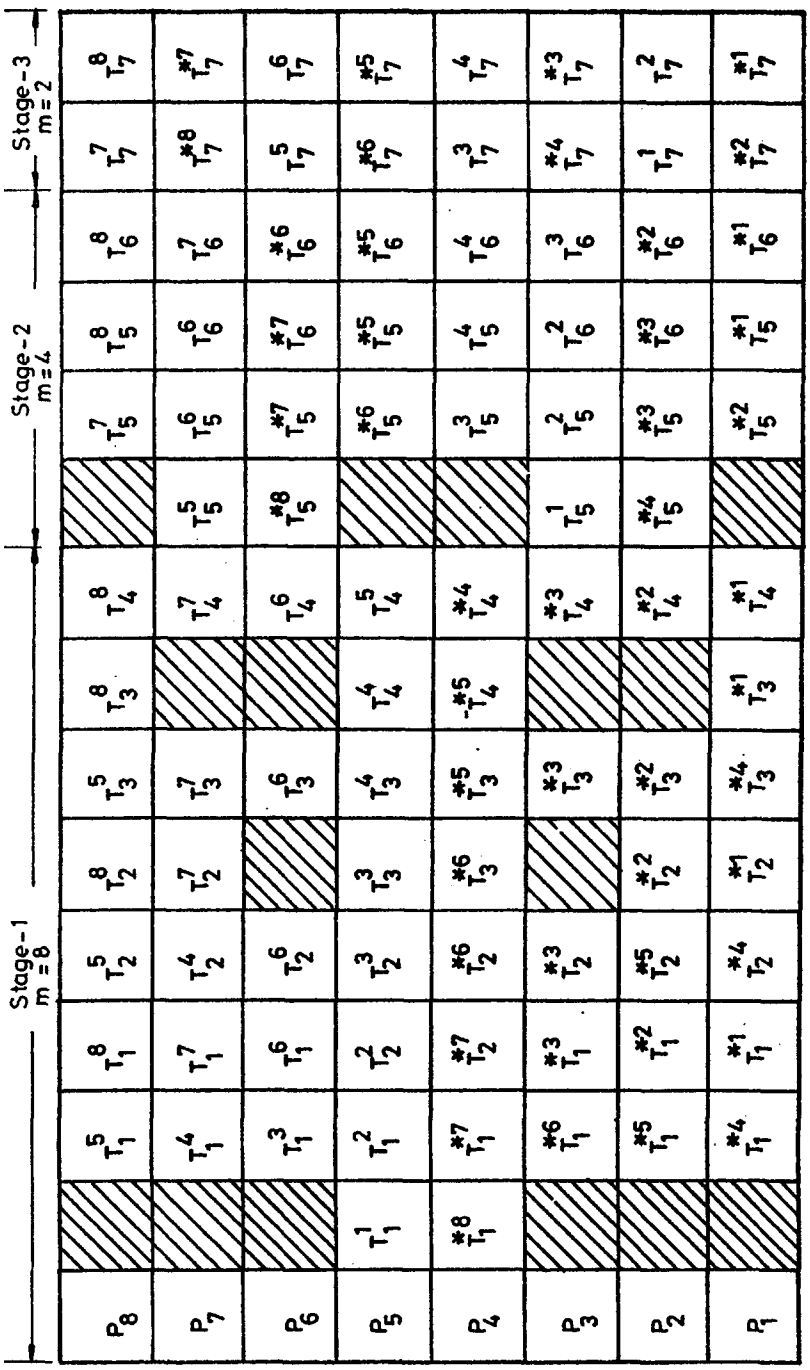

Figure 7. Complete schedule of tasks in the SGE algorithm to produce the diagonal form for $N = 8$.

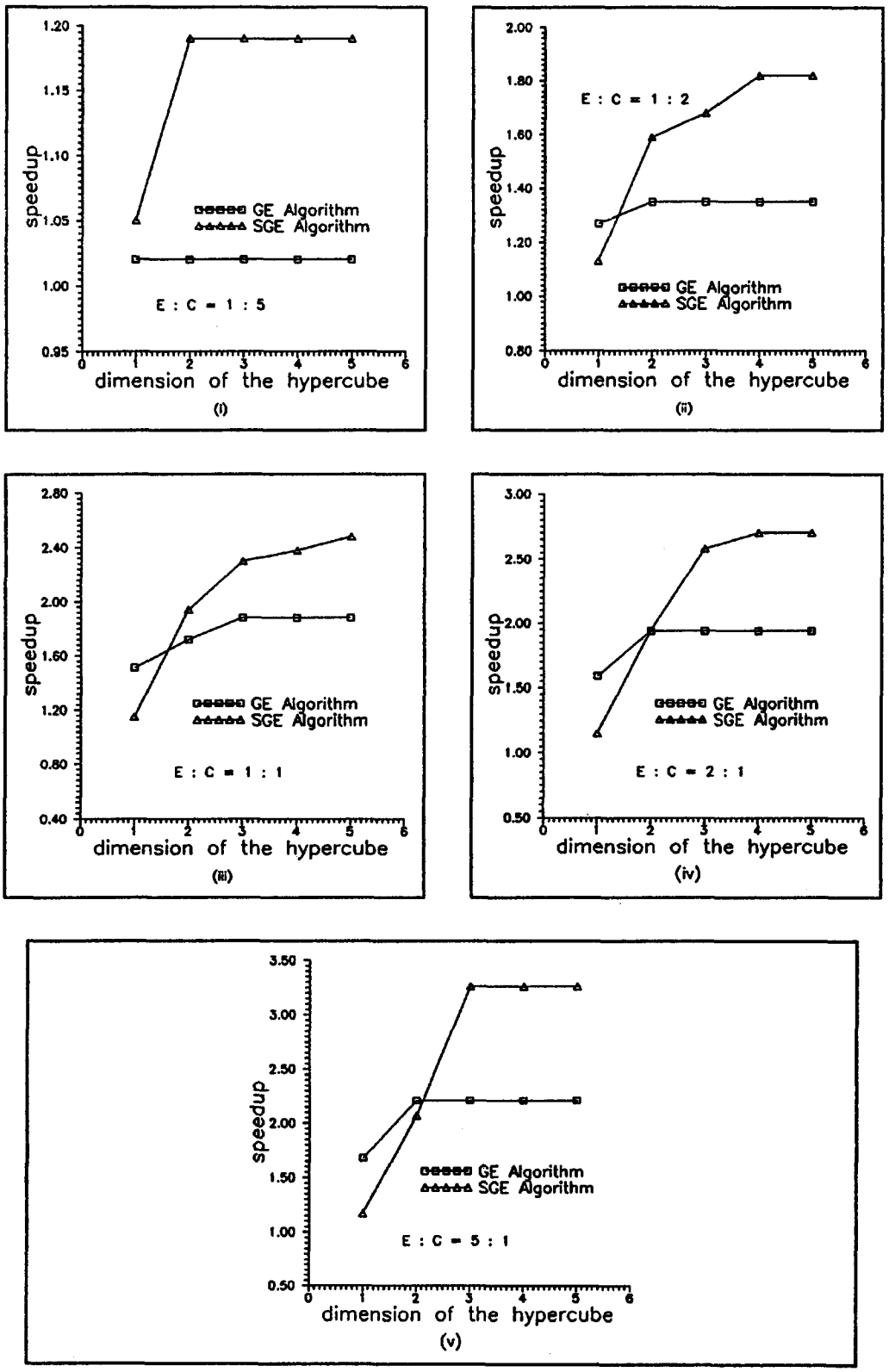

Figure 8. Performance of the algorithms for $N = 4$.

6. PERFORMANCE EVALUATION OF THE SGE ALGORITHM

The scheduling scheme described above assumes that N processors are available on the given multiprocessor system for solving a system of N linear equations. Further, it does not take the intertask (interprocessor) communication times into account. Now using an efficient scheduling

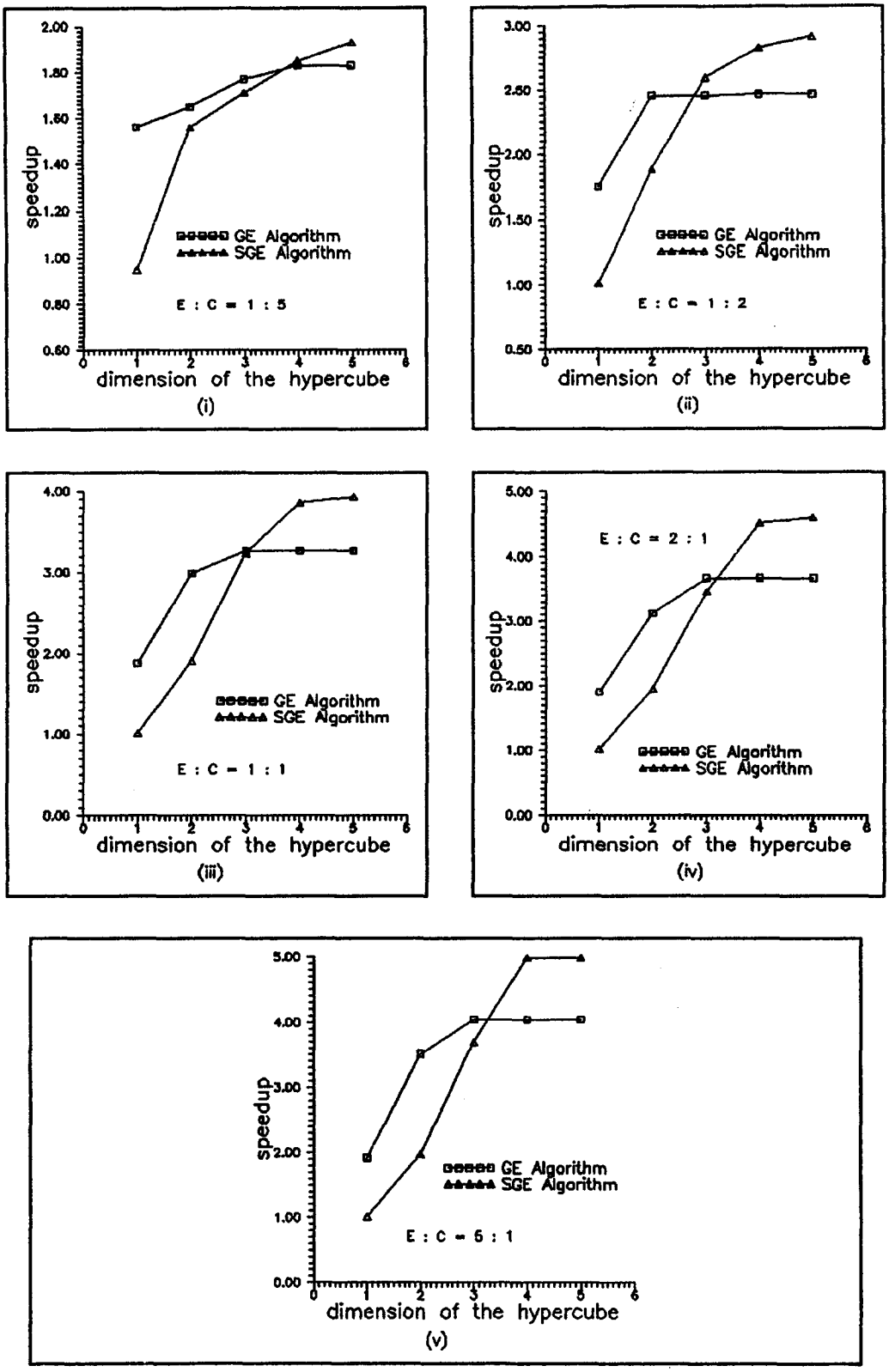

Figure 9. Performance of the algorithms for $N = 8$.

algorithm, for assigning the tasks of a precedence-constrained task graph onto the processors of a multiprocessor system with arbitrary topology considering nonnegligible intertask communication times, we evaluate the performance of the SGE algorithm. The scheduling algorithm [13] follows the basic list scheduling theory. It is designed to exploit the schedule holes in both the processor

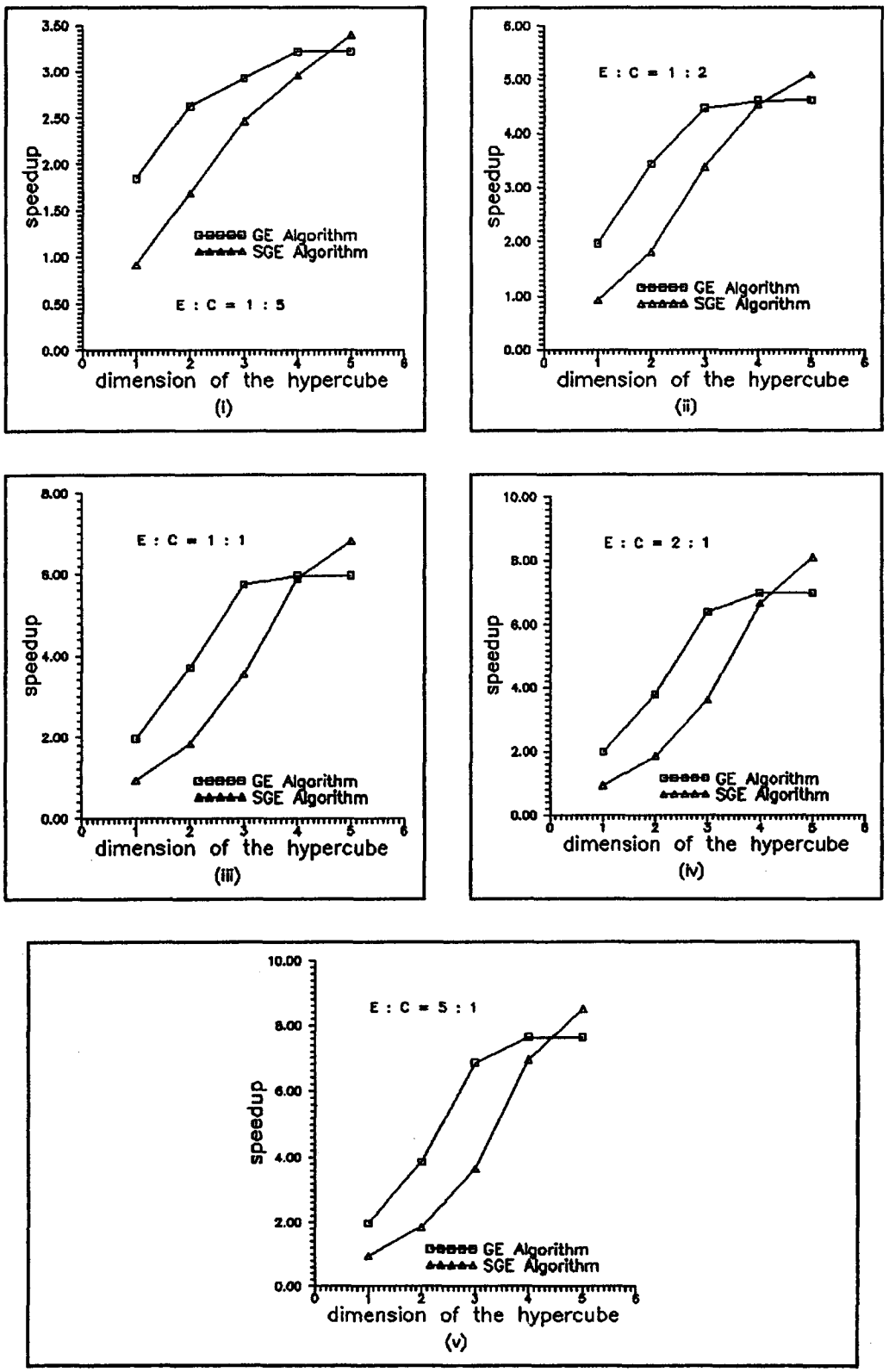

Figure 10. Performance of the algorithms for $N = 16$.

and the link schedules. Further, the scheduling algorithm [13] produces contention-free schedules. Interested readers are referred to [13] for the details of the scheduling algorithm.

In order to evaluate the performance of the SGE algorithm, we conducted the following experiments on a network of SUN workstations.

(1) We generated task graphs for both GE and SGE algorithms for various problem sizes, and scheduled them onto a hypercube multiprocessor system using the scheduling algorithm [13]. We assumed that each floating point arithmetic operation (E) takes one unit of time and the transfer of a floating point number between two adjacent processors (C) takes 5, 2, 1, 1/2, and 1/5 units of time for the purpose of generating the task schedules.

(2) We implemented the task schedules obtained above on a network of SUN workstations using P4 (Portable Programs for Parallel Processors) software for verifying the correctness of the algorithms. P4, developed at Argonne National Laboratory, is a library of macros and subroutines for programming a variety of parallel machines including a network of workstations.

The speedup of a parallel algorithm is measured as the ratio of the completion time of a task graph of the sequential GE algorithm on one processor and the completion time of the task graph of the parallel algorithm on a hypercube multiprocessor system. We have presented the results obtained with the two algorithms as plots of speedup versus dimension of the hypercube for different values of $\langle E : C, N \rangle$ in Figures 8–10. From these graphs, we can clearly see that the speedup obtained by using the SGE algorithm is always greater than the speedup of the GE algorithm when the number of processors is greater than the number of equations in the system. This increase in speedup is due to the fact that the SGE algorithm has higher degree of inherent parallelism and the intertask communication becomes more localized as the algorithm progresses (see Figure 6 for $N = 8$).

7. CONCLUSIONS

We have presented a new parallel algorithm called Successive Gaussian Elimination (SGE) for the solution of linear equations.

The main features of the SGE algorithm are:

(a) It produces diagonal form in $O(N^2)$ time steps using $O(N)$ processors against the same number of time steps and processors required for producing the triangular form in the existing methods.

(b) The back substitution phase, which takes $O(N)$ steps using $O(N)$ processors or $O\left(\log_2^2 N\right)$ steps using $O(N^3)$ processors [2,4], is completely replaced by one step division in the SGE algorithm.

(c) The algorithm supports partial pivoting to improve numerical stability.

(d) The SGE algorithm permits pivot column tasks to be executed in parallel with nonpivot column tasks. Our scheduling strategy cleverly exploits this parallelism in the algorithm using minimum number of processors.

(e) In the SGE algorithm, all x_i $(i = 1, 2, \ldots, N)$ are found simultaneously, unlike in the conventional GE algorithm in which x_i $(i = N, N-1, \ldots, 2)$ is used for finding x_j $(j = 1, 2, \ldots, i-1)$. Hence, the SGE algorithm is expected to have better numerical stability characteristics than the conventional GE algorithm.

REFERENCES

1. D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computations—Numerical Methods*, Prentice Hall, New Jersey, (1989).
2. D. Heller, A survey of parallel algorithms in numerical linear algebra, *SIAM Review* **20** (4), 740–777 (October 1978).
3. S. Lakshmivarahan and S.K. Dhall, *Analysis and Design of Parallel Algorithms—Arithmetic and Matrix Problems*, McGraw-Hill, New York, (1990).
4. A.H. Sameh and D.J. Kuck, Parallel direct linear system solvers—A survey, In *Parallel Computers—Parallel Mathematics*, (Edited by Feilmeier), pp. 25–30, (1977).

5. E. Chu and A. George, Gaussian elimination with partial pivoting and load balancing on a multiprocessor, *Parallel Computing* **5** (1/2), 65–74 (July 1987).
6. K.A. Gallivan, R.J. Plemmons and A.H. Sameh, Parallel algorithms for dense linear algebra computations, *SIAM Review* **32** (1), 54–135 (March 1990).
7. G.H. Golub and C.F. Van Loan, *Matrix Computations*, John Hopkins University Press, (1989).
8. R.E. Lord, J.S. Kowalik and S.P. Kumar, Solving linear algebraic equations on a MIMD computer, *Journal of ACM* **30** (1), 103–117 (January 1983).
9. M. Veldhorst, Gaussian elimination with partial pivoting on an MIMD computer, *Journal of Parallel and Distributed Computing* **6** (1), 62–68 (February 1989).
10. R. Melhem, Parallel Gauss-Jordan elimination for solution of dense linear equations, *Parallel Computing* **4** (3), 339–343 (June 1987).
11. A.H. Sameh and D.J. Kuck, On stable parallel linear system solvers, *Journal of ACM* **25** (1), 81–91 (January 1978).
12. M.K. Sridhar, A new algorithm for parallel solution of linear equations, *Information Processing Letters* **24**, 407–412 (1987).
13. C. Siva Ram Murthy, K.N. Balasubramanya Murthy and A. Srinivas, Scheduling of precedence-constrained parallel program tasks on multiprocessors, *Microprocessing and Microprogramming* **36** (2), 93–104 (March 1993).