# An Improved Mapping of Cyclic Elimination onto Hypercubes using Data Replication

Kartik Gopalan C. Siva Ram Murthy \*

K. N. Balasubramanya Murthy

Department of Computer Science and Engineering

Indian Institute of Technology

Madras 600 036, India

e-mail: murthy@iitm.ernet.in

#### Abstract

In this paper, we propose a new mapping of the Cyclic Elimination (CE) algorithm for the solution of block tridiagonal linear system of equations onto hypercube multiprocessors. Unlike the previous mapping schemes, in our mapping of the CE algorithm all communications are restricted to physically adjacent processors, using the concept of data replication. The effectiveness of our mapping is demonstrated by comparing it with the existing mapping of the Cyclic Reduction algorithm onto hypercubes using both analytical and simulation methods.

**Key Words:** Linear Algebra, Block Tridiagonal Linear Systems, Cyclic Elimination, Cyclic Reduction, Hypercube Multiprocessors.

## 1 Introduction

The numerical solution of block tridiagonal linear system of equations is one of the important classes of problems which occurs in many areas of numerical analysis such as solving partial differential equations using finite difference schemes. The most efficient method for solving block tridiagonal linear systems on a uniprocessor is the Block Gaussian Elimination (BGE) [17]. However, the BGE algorithm is not suitable for multiprocessor environment

<sup>\*</sup>Author for correspondence.

because of lack of adequate parallelism. On the other hand algorithms such as block Cyclic Reduction(CR) [3], Buneman's algorithm [15], block Cyclic Elimination(CE) [16, 17] and recursive doubling [4] exploit the inherent parallelism present in the problem. For efficient implementation of these algorithms on multiprocessors, the principal challenge lies in reducing the overhead involved in communication between processors. This aim can be achieved by using efficient mapping schemes and overlapping the communication and computation steps.

A mapping of any algorithm onto a hypercube is said to be *desirable* if all communications are restricted to physically adjacent processors. However, the following (statement) result due to Lakshmivarahan and Dhall [4] relates to non-existence of a desirable mapping of the CR and CE algorithms onto base-2 (binary) hypercube.

"In any mapping of the CR or CE algorithm onto a p-node base-2 hypercube, it is necessary that at least  $\frac{\log p}{2} - 1$  steps involve communication between processors that are at a distance two or more apart." (For proof refer to [4], pp 364-365.) Further, it has been shown by Johnsson [18] that upon using the binary reflected Gray code mapping [9], the distance between any two communicating processors is no more than two. However, we show, in this paper, that it is possible to obtain a desirable mapping of CE algorithm onto hypercube multiprocessors using the concept of data replication.

Complete details about mapping of CR or CE algorithm onto a hypercube multiprocessor can be found in [4]. Here we give a brief overview of the major differences between the CR and CE algorithms. The CR algorithm consists of two phases - reduction and substitution. The CE algorithm consists of only one phase, namely, reduction. The degree of parallelism in the reduction phase of CR algorithm halves with every consecutive stage. On the other hand, the degree of parallelism in the reduction phase of CE algorithm remains constant through all stages. Thus, theoretically, CE algorithm ought to be preferred over CR algorithm. However, the communication overhead incurred in the existing mapping of CE algorithm onto hypercubes is much higher than that of CR algorithm. In particular, the communication graph of the CR algorithm is a sub-graph of the communication graph of CE algorithm. The communication overhead incurred by the existing mapping of CE algorithm becomes costly, especially since successive stages of the reduction phase call for data communication between processors which are not neighbours. A large number of such multiple hop data communications lead to link contentions and, consequently, lower performance.

In order to gainfully exploit the higher degree of parallelism of the CE algorithm we propose an improved mapping of the CE algorithm onto a hypercube multiprocessor with which the data communications are restricted to occur between neighbouring processors only. This is achieved by efficient duplication of data at every stage of the algorithm. Thus the problem due to link contentions are overcome and better performance achieved. To the best of our knowledge, the existing literature does not contain any algorithm which solves the block tridiagonal linear systems on a hypercube using only neighbouring processor communication. Two significant features of our algorithm are that, the computational load is balanced among all the processors at all stages of the algorithm and secondly, much of the communication gets overlapped with computation giving an overall better performance.

The rest of the paper is organised as follows. In section 2, we make a problem statement and introduce some notations which will be used in the subsequent sections. In section 3, we discuss the sequential BGE algorithm on a uniprocessor, and the parallel CR and CE algorithms. In section 4, using an example, we first look at the existing schemes for mapping CR and CE algorithms onto hypercube multiprocessors and then present our improved mapping scheme for the CE algorithm followed by its analytical performance study. In section 5, we present numerical results for the speedups obtained from our new mapping scheme and the existing mapping of CR algorithm, and compare the two schemes. Section 6 concludes the work with some pointers for future research.

## 2 Problem Statement and Notations

The block tridiagonal matrix A is defined as

$$A = \begin{pmatrix} d_1 & f_1 \\ e_2 & d_2 & f_2 \\ & \ddots & \ddots & \ddots \\ & & e_{N-1} & d_{N-1} & f_{N-1} \\ & & & e_N & d_N \end{pmatrix}$$

where the components  $e_i$ ,  $d_i$  and  $f_i$  are  $n \times n$  matrices (or blocks) with  $e_1 = f_N = 0$ . There are N such blocks along principal diagonal of A where N is a power of 2. So the overall dimension of A is  $(Nn) \times (Nn)$ . We are to solve the system AX = v, where the vector  $X = (x_1, x_2, \ldots, x_N)^t$ , the vector  $v = (v_1, v_2, \ldots, v_N)^t$ , the components  $x_i$  and  $v_i$  are n-vectors

and

$$e_j x_{j-1} + d_j x_j + f_j x_{j+1} = v_j$$
,  $j = 1, ..., N$ .

The CR algorithm for solving the system Ax = v consists of the reduction phase followed by the back substitution phase. Each of these two phases, in turn, is divided into  $\log N$  stages. The CE algorithm consists of reduction phase alone which is divided into  $\log N$  stages. In both CR and CE algorithms, at the beginning of stage l = 1 of the reduction phase, we define the 5-tuple  $row_i^{(0)}$  as

$$row_i^{(0)} = (e_i^{(0)}, d_i^{(0)}, (d_i^{(0)})^{-1}, f_i^{(0)}, v_i^{(0)}) = (e_i, d_i, (d_i)^{-1}, f_i, v_i).$$

At each stage  $l \in \{1, ..., \log N\}$  of reduction phase, we define the tuple  $row_i^{(l)}$  as

$$row_i^{(l)} = \begin{cases} (e_i^{(l)}, d_i^{(l)}, (d_i^{(l)})^{-1}, f_i^{(l)}, v_i^{(l)}) &, \forall i \in \{1, \dots, N\} \\ (0, I, I, 0, 0) &, \forall i \leq 0 \text{ or } i > N. \end{cases}$$

Here  $e_i^{(l)}$  is the value of  $e_i$  at the end of stage l,  $f_i^{(l)}$  is the value of  $f_i$  at the end of stage l and so on. The matrix I is the  $n \times n$  identity matrix. Note that  $(d_i^{(l)})^{-1}$  is included as a member of the tuple  $row_i^{(l)}$ . This is done because, the inverse, once computed at a source processor, can be transferred along with the tuple  $row_i^{(l)}$  to other processors which need it, thus avoiding its re-computation at the destination processors. Figure 1 gives an example of the above notations for an  $8 \times 8$  block tridiagonal system.

$$\begin{pmatrix} d_1 & f_1 & & & & \\ e_2 & d_2 & f_2 & & & \\ & \ddots & \ddots & \ddots & \\ & & e_7 & d_7 & f_7 \\ & & & e_8 & d_8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_7 \\ v_8 \end{pmatrix}$$

$$A \qquad x = v$$

Figure 1: An  $8 \times 8$  block tridiagonal system and listing of  $row_i^{(l)}$  at various stages

## 3 Solving Block Tridiagonal Linear Systems

In this section, we first briefly present the theoretical concepts behind the sequential BGE and then the parallel versions of CR and CE algorithms.

## 3.1 Sequential Block Gaussian Elimination (BGE)

There are two phases in this algorithm - forward elimination and back substitution. Computation within each phase is completely sequential in nature.

```
Algorithm 1
(*Forward elimination phase*)
for i = 2 to N do
    Calculate (d_{i-1})^{-1}
    a_i = e_i(d_{i-1})^{-1}
    e_i = 0
    d_i = d_i - a_i f_{i-1}
    f_i = fi
    v_i = v_i - a_i v_{i-1}
endfor
(*Back substitution phase*)
Calculate (d_N)^{-1}
x_N = (d_N)^{-1} v_N
for i = (N-1) downto 1 do
    x_i = (d_i)^{-1}(v_i - f_i x_{i+1})
endfor.
```

The time taken for calculating the inverse of an  $n \times n$  matrix block, using the exchange method, is  $T_{inv} = 3n^3 - 4n^2 + 2n$  computational time units. Multiplying two  $n \times n$  matrices takes  $T_{mult} = 2n^3 - n^2$  time units, whereas multiplying an  $n \times n$  matrix with an n-vector takes  $T'_{mult} = 2n^2 - n$  time units. The sequential BGE algorithm executes N matrix inversions, 2(N-1) matrix-matrix multiplications, 3N-2 matrix-vector multiplications, N-1 matrix subtractions, and 2(N-1) vector subtractions. Summing up all the components, this step takes

$$T_{BGE} = N(3n^3 - 4n^2 + 2n) + 2(N - 1)(2n^3 - n^2) + (3N - 2)(2^2 - n) + (N - 1)n^2 + 2(N - 1)n$$
$$= (N - 1)(7n^3 + n^2 + n) + (3n^3 + 2n^2 + n) \text{ time units.}$$

## 3.2 The Basic Elimination Step

Both CE and CR algorithms, have a basic elimination step in common. We name this step  $Compute\ row_i^{(l)}$ , where  $i\in\{1,\ldots,N\}$  is the index of a row of blocks and  $l\in\{1,\ldots,\log N\}$  is the stage being considered. Let  $h=2^{(l-1)}$ . The  $Compute\ row_i^{(l)}$  step eliminates the dependence of equation i on the variables  $x_{i+h}$  and  $x_{i-h}$  by subtracting appropriate multiples of equations i+h and i-h from equation i. The  $Compute\ row_i^{(l)}$  step consists of the following computation steps.

$$\begin{split} a_i^{(l)} &= -e_i^{(l)} (d_{i-h}^{(l-1)})^{-1} \\ b_i^{(l)} &= -f_i^{(l)} (d_{i+h}^{(l-1)})^{-1} \\ e_i^{(l)} &= a_i^{(l)} e_{i-h}^{(l-1)} \\ d_i^{(l)} &= a_i^{(l-1)} + a_i^{(l)} f_{i-h}^{(l-1)} + b_i^{(l)} e_{i+h}^{(l-1)} \\ \text{Calculate } (d_i^{(l)})^{-1} \\ f_i^{(l)} &= b_i^{(l)} f_{i+h}^{(l-1)} \\ v_i^{(l)} &= v_i^{(l-1)} + a_i^{(l)} v_{i-h}^{(l-1)} + b_i^{(l)} v_{i+h}^{(l-1)} \end{split}$$

The Compute  $row_i^{(l)}$  step involves six matrix-matrix multiplications, two matrix-vector multiplications, one matrix inversion, two matrix additions, and two vector additions. Summing up the components, this step takes  $e = (15n^3 - 4n^2 + 2n)$  time units.

## 3.3 The Block Cyclic Reduction Algorithm (CR)

The CR algorithm consists of two phases, namely reduction (or elimination) phase and back substitution phase. These two phases are essentially sequential although the computations within each phase can be carried out in parallel. Therefore, the total parallel time is the sum of the individual parallel times. Figure 2 shows the pattern of elimination and back substitution steps for the case of N=8 block equations.

```
Algorithm~2\\ (*Reduction~phase*)\\ 1.~\mathbf{for}~l=1~\mathbf{to}~\log N~\mathbf{do}\\ h=2^{(l-1)}\\ \mathbf{for}~i\in\{2^l,2\times 2^l,3\times 2^l,\dots,\log N\}\mathbf{do}~\mathbf{in}~\mathbf{parallel}\\ Compute~row_i^{(l)}\\ \mathbf{endfor}\\ \mathbf{endfor}\\
```

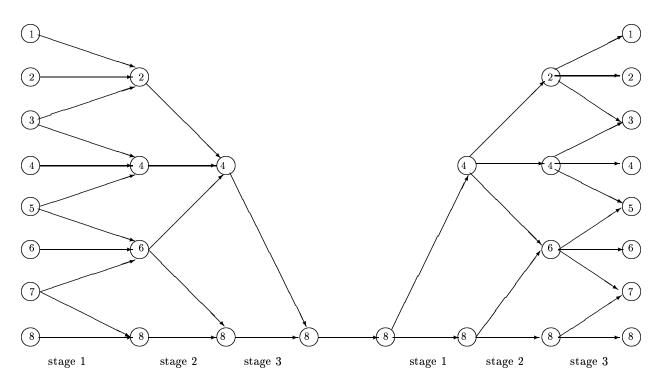


Figure 2: Elimination and back substitution pattern in CR algorithm for N=8

```
(*Back substitution phase*)

2. x_N = (d_N^{(\log N)})^{-1} v_N^{\log N}

3. for l = \log N downto 1 do

h = 2^{(l-1)}

for i \in \{2^{(l-1)}, 3 \times 2^{(l-1)}, 5 \times 2^{(l-1)}, \dots, N-2^{(l-1)}\}do in parallel x_i = (d_i^{(l-1)})^{-1} (v_i^{(l-1)} - e_i^{(l-1)} x_{i-h} - f_i^{(l-1)} x_{i+h}) endfor endfor.
```

## 3.4 The Block Cyclic Elimination Algorithm (CE)

The CE algorithm consists of only the elimination phase followed by a single step division. Here the elimination phase recursively converts the given system of equations into two independent systems of equations each of which can be solved in parallel using the CE algorithm. Figure 3 shows the pattern of  $Compute\ row_i^{(l)}$  steps for the case of N=8 block equations.

Algorithm 3 1. for l = 1 to  $\log N$  do

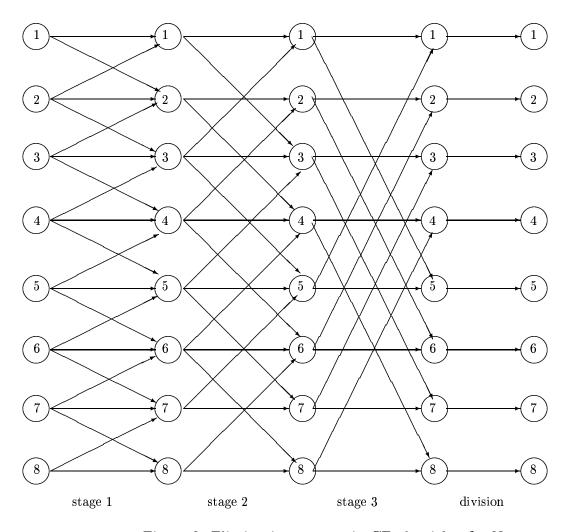


Figure 3: Elimination pattern in CE algorithm for N=8

```
h=2^{(l-1)} for i\in\{1,2,\ldots,N\}do in parallel Compute\ row_i^{(l)} endfor endfor i\in\{1,2,\ldots,N\}do in parallel x_i=(d_i^{(\log N)})^{-1}v_i^{(\log N)} endfor.
```

## 4 Solving Block Tridiagonal Linear Systems on Hypercubes

The hypercube, one of the most popular architecture for multiprocessor systems, is a generalization of a cube to d dimensions such that each of the  $2^d$  processors has d neighbours. In this section, we present an improved mapping of the CE algorithm on a hypercube multiprocessor which achieves neighbouring processor communication by efficient use of the concept of data duplication. We begin by comparing the three mapping schemes, namely, the existing mapping of the CR algorithm, the existing mapping of the CE algorithm, and our improved mapping of the CE algorithm with the help of a simple example. We then proceed to formally present our algorithm and explain the various steps.

## 4.1 Comparison of the Three Schemes

Let us consider the simple problem of solving a block tridiagonal system with N=16 block equations and block size  $1 \times 1$  (i.e., n=1) on a two-dimensional hypercube (i.e., there are four processors in the hypercube). We trace the step by step execution of each of the schemes below and calculate the time taken in each case. For the sake of simplicity, we consider only the non-overlapped execution of computation and communication steps.

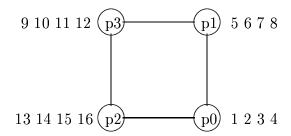
We define the following notations to make our comparison clearer.

- $p_k$  symbolically represents the kth processor of a hypercube.
- p represents the number of processors in a hypercube. Thus the dimension of the hypercube is  $\log p$ .

- e represents the number of operations involved in executing the  $Compute\ row_i^{(l)}$  with no communication overheads. As shown in section 3.2, this works out to be  $e = 15n^3 4n^2 + 2n$  computational time units.
- s represents the number of operations involved in executing one back substitution step, which involves three matrix-vector multiplications and two vector subtractions. This works out to be  $s = 6n^2 n$  computational time units.
- Communication to Computation ratio, C/E, represents the the ratio of time taken to communicate one floating point value between two neighbouring processors to the time taken to execute one floating point operation.
- $T_b$  represents the time taken to communicate the contents of an  $n \times n$  matrix block between two neighbouring processors. This works out to be  $n^2(C/E)$  computational time units.
- $T_e$  represents the time taken to communicate the contents of a 5-tuple  $row_i^{(l)}$  between two neighbouring processors. This works out to be  $5T_b = 5n^2(C/E)$  computational time units.
- $kth\ dimension$  of a hypercube is represented by a set of links each of which connects some processor  $p_j$  to its neighbour  $p_{j'}$ , such that j' is obtained by inverting the kth bit in the binary representation of j.

#### 4.1.1 Existing Mapping of the CR Algorithm

Figure 4 shows the step by step execution of the CR algorithm for solving the tridiagonal system of 16 equations using a hypercube of four processors. The equations are initially mapped onto processors in a block wrap manner (see figure 4(a)). The reduction phase of the mapped algorithm consists of 4 (i.e., log 16) stages. The first stage consists of a one hop communication of tuples  $row_5^{(0)}$  (from processor  $p_1$  to  $p_0$ ),  $row_9^{(0)}$  (from  $p_3$  to  $p_1$ ),  $row_{13}^{(0)}$  (from  $p_2$  to  $p_3$ ) followed by the computation steps  $Compute\ row_2^{(1)}$  and  $Compute\ row_{12}^{(1)}$  at  $p_0$ ,  $Compute\ row_{12}^{(1)}$  and  $Compute\ row_{12}^{(1)}$  at  $p_1$ ,  $Compute\ row_{10}^{(1)}$  and  $Compute\ row_{12}^{(1)}$  at  $p_3$  and  $Compute\ row_{14}^{(1)}$  and  $Compute\ row_{16}^{(1)}$  at  $p_2$ . This completes the first stage of reduction phase. Similarly, second and third stages involve one hop communication of  $row_i^{(l)}$  tuples and one step each of the form  $Compute\ row_i^{(l)}$ . Stage 4 consists of a two hop communication of  $row_8^{(l)}$  from  $p_1$  to  $p_2$  followed by the step  $Compute\ row_{16}^{(l)}$ . The substitution phase of the algorithm follows a completely reverse pattern of communication and can be described by reversing the order of the stages and the direction of the arrows in the reduction phase. The



## (a) initial data distribution

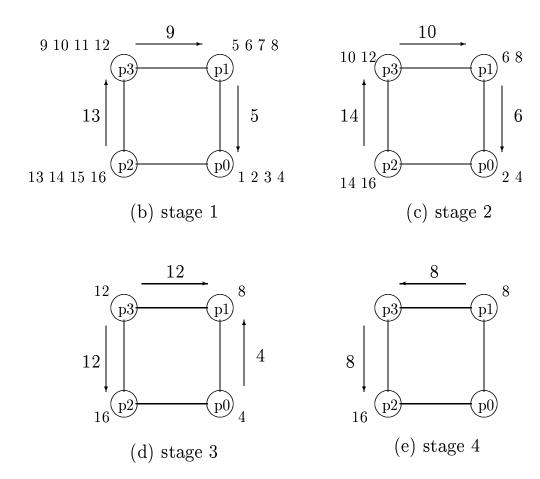


Figure 4: Progression of the CR algorithm with the existing mapping for N=16 and p=4

Table 1: Counts of tasks executed by the CR algorithm

		task count	
	stage	$T_e$	e
Poduction phase	1	1	2
Reduction phase	2	1	1
	3	1	1
	4	2	1

Substitution	phase
--------------	-------

		task count		
	stage	$T_b$	s	
	1	2	1	
•	2	1	1	
	3	1	1	
	4	1	2	

data items communicated are the floating point values of the variables  $x_i$  (instead of  $row_i^{(l)}$  as in reduction phase).

The counts of various tasks executed at each stage of the algorithm are summarised in table 1. We see from table 1 that it takes  $5T_e + 5e$  computational time units for the reduction phase, followed by a division step, followed by  $5T_b + 5s$  units for the substitution phase. Thus the total execution time is  $T_{CR} = 5(T_e + T_b) + 5(e + s) + 1$  units. Typically the communication to computation ratio (C/E) is of the order of 100. Thus with N = 16, n = 1 and p = 4 we have  $T_e = 500$ ,  $T_b = 100$ , e = 13 and s = 5. Thus  $T_{CR} = 3091$  computational time units from the above expression.

#### 4.1.2 Existing Mapping of the CE Algorithm

Figure 5 shows the step by step execution of the CE algorithm for solving the tridiagonal system of 16 equations using a hypercube with four processors. The equations are initially mapped onto processors in a block wrap manner (see figure 5(a)). The algorithm consists of only reduction phase which has 4 (i.e.,  $\log 16$ ) stages. In the first stage,  $row_5^{(0)}$  tuple is communicated from  $p_1$  to  $p_0$  preceding the step  $Compute\ row_4^{(1)}$ . Simultaneously,  $row_4^{(0)}$  tuple is communicated from  $p_0$  to  $p_1$  preceding the step  $Compute\ row_5^{(1)}$  and so on. Thus stage 1 consists of one-hop communication of  $row_i^{(l)}$  tuples followed by four  $Compute\ row_i^{(l)}$  steps per processor. At the end of stage 1, there are two independent sets of equations, namely,  $\{1, 3, 5, 7, 9, 11, 13, 15\}$  and  $\{2, 4, 6, 8, 10, 12, 14, 16\}$ . Similarly, stage 2 consists of two one-hop communication of  $row_i^{(l)}$  tuples followed by four  $Compute\ row_i^{(l)}$  steps per processor. At the end of stage 2 there are four independent sets of equations, namely  $\{1, 5, 9, 13\}$ ,  $\{3, 7, 11, 15\}$ ,  $\{2, 6, 10, 14\}$ , and  $\{4, 8, 12, 16\}$ . Stage 3 consists of four one hop communications of  $row_i^{(l)}$  tuples followed by four  $Compute\ row_i^{(l)}$  steps.

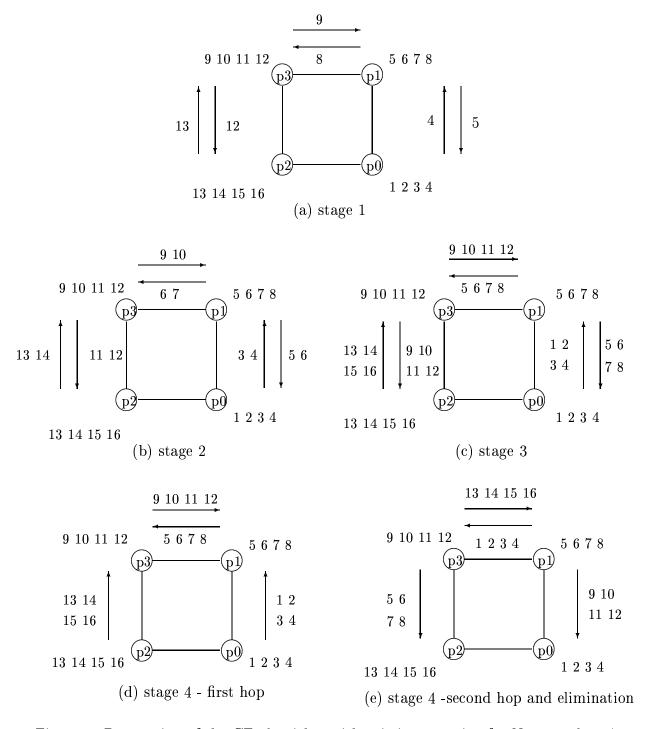


Figure 5: Progression of the CE algorithm with existing mapping for N=16 and p=4

Table 2: Counts of tasks executed by the CE algorithm with the existing mapping

	task count	
stage	$T_e$	e
1	1	4
2	2	4
3	4	4
4	8(2hops)	4

The counts of various tasks executed at each stage of the algorithm are summarised in table 2. We see from table 2 that communication overhead doubles with each stage as the number of independent sets of equations doubles at each stage. Further, the last stage consists of four consecutive two-hop communication of  $row_i^{(l)}$  tuples. Stage 4 is followed by four divisions per processor. Thus the total execution time taken in the present case is  $T_{CE} = 15T_e + 16e + 4$  computational time units. Substituting the values for  $T_e$  and e, we get  $T_{CE} = 7712$  time units. Thus, in the present case, the existing mapping of CE algorithm performs poorly in comparison to the mapping of CR algorithm onto hypercubes.

#### 4.1.3 The Improved Mapping of CE Algorithm

Figure 6 shows the step by step execution of our improved mapping of CE algorithm for solving the tridiagonal system of 16 equations using a hypercube with four processors. In this improved mapping scheme, all the communication steps occur between neighbouring processors only. The initial distribution of data is as follows. We divide the processors of the hypercube into two sets -  $\{p_0, p_1\}$  and  $\{p_2, p_3\}$  - the former being the set of processors in the lower half of the hypercube along 2nd dimension and the latter being the set of processors in the upper half of the hypercube along the 1st dimension. The 16 equations are then mapped onto each of the two sets of processors in a block wrap manner. Thus we get the initial data distribution as shown in figure 6(a). There are  $\log p$  stages of the improved mapping. Each of the first  $\log p - 1$  stages (only the first stage in the present case) consists of two phases - elimination and replication (copying). The elimination phase corresponds to the reduction stage of the CE algorithm in which Compute  $row_i^{(l)}$  steps are executed. Thus in stage 1 of the algorithm (figure 6(b)), the processors in the set  $\{p_0, p_1\}$  execute Compute  $row_i^{(l)}$  steps for odd-indexed equations and the processor set  $\{p_2, p_3\}$  executes Compute  $row_i^{(l)}$  for even-indexed equations. This involves a one-hop communication of  $row_i^{(l)}$  tuples followed by

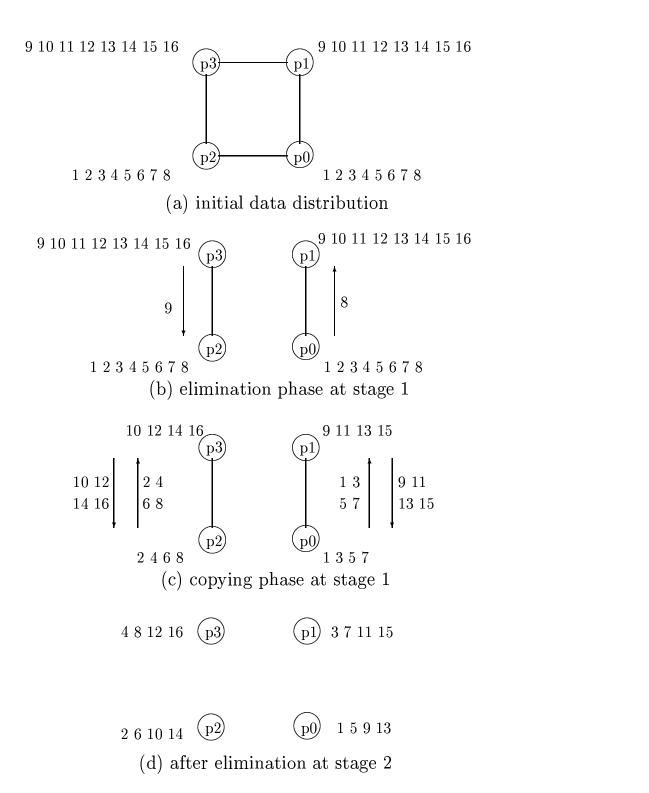


Figure 6: Progression of the CE algorithm with improved mapping for N=16 and p=4  $\,$ 

Table 3: Counts of tasks executed by the CE algorithm with improved mapping

	stage	task count	
		$T_e$	e
1	elimination	1	4
	copying	4	0
2	elimination	0	4

four  $Compute\ row_i^{(1)}$  steps per processor. At the end of elimination phase of stage 1, the processor set  $\{p_0, p_1\}$  holds the independent set of equations  $\{1, 3, 5, 7, 9, 11, 13, 15\}$  and the processor set  $\{p_2, p_3\}$  holds the independent set of equations  $\{2, 4, 6, 8, 10, 12, 14, 16\}$ . The next phase of stage 1 is the  $copying\ phase$  in which each processor copies the  $row_i^{(1)}$  tuples of its set of equations to the neighbouring processor along the 1st dimension of the hypercube. Thus  $p_0$  copies the  $row_i^{(1)}$  tuples of equations  $\{1, 3, 5, 7\}$  to  $p_1$  and  $p_1$  copies those of equations  $\{9, 11, 13, 15\}$  to  $p_0$ . Similar copying occurs between processors  $p_2$  and  $p_3$ . Stage 2 of the algorithm consists of only the elimination phase. Thus  $p_0$  executes  $Compute\ row_i^{(2)}$  steps for  $i = 1, 5, 9, 13,\ p_1$  executes  $Compute\ row_i^{(2)}$  steps for  $i = 3, 7, 11, 15,\ p_2$  executes  $Compute\ row_i^{(2)}$  steps for i = 2, 6, 10, 14, and  $p_3$  executes  $Compute\ row_i^{(2)}$  steps for i = 4, 8, 12, 16. Thus at the end of  $\log p$  stages (i.e., elimination phase of stage 2 in the present case) each processor contains an independent set of equations which can be solved using BGE algorithm without communicating with any other processor.

The counts of various tasks executed at each stage of the algorithm are summarised in table 3. The BGE algorithm for solving 4 equations per processor takes  $T_{BGE}=33$  computational time units (see section 3.1). Thus the total time taken in the present case is  $T_{new}=5T_e+8e+T_{BGE}$  units. Substituting the values for  $T_e$  and e, we get  $T_{new}=2637$  time units.

Thus we see that in the case of N=16, n=1 and p=4, our improved mapping of CE algorithm performs better than the existing mappings of both CR and the CE algorithms. Further, the existing mapping of CR algorithm performs better than the existing mapping of CE algorithm due to lower communication overhead. We now present some definitions and then formally present our improved mapping of the CE algorithm. We then evaluate its performance by comparing with the existing mapping of the CR algorithm only, since this mapping fares better than the existing mapping of CE algorithm, as shown in the above example.

#### 4.2 Definitions

• Binary reflected gray codes [9] are a class of codes useful in embedding a ring structure onto a binary hypercube. Let G(n) denote the set of all n-digit code words of the base-2 (binary) reflected gray code i.e.,

$$G(n) = \{G_0(n), G_1(n), \dots, G_{2^n-1}(n)\}\$$

where,  $G_i(n)$  ith code word of binary reflected gray code,  $i \in \{0, \ldots, 2^n - 1\}$ . Let

$$i = i_n i_{n-1} \cdots i_2 i_1 i_0$$

in binary with  $i_n = 0$  and

$$G_i(n) = g_n g_{n-1} \cdots g_2 g_1$$

in binary. If  $\oplus$  denotes the exclusive-OR addition of binary bits, then the encoding function  $E_n :< \mathcal{N} > \rightarrow G(n)$  is given by

$$E_n(i) = G_i(n) = g_n g_{n-1} \cdots g_1$$

where

$$g_j = i_j \oplus i_{j-1}$$

for all j = 1, 2, ..., N, and the decoding function  $D_n : G(n) \to < \mathcal{N} >$  is given by

$$D_n(q) = i$$

where

$$i_j = g_{j+1} \oplus g_{j+2} \oplus \cdots \oplus g_n.$$

- $p_j$ :  $send(row_i^{(l)}, p_{j'})$  indicates that processor  $p_j$  sends contents of  $row_i^{(l)}$  to processor  $p_{j'}$ .
- $p_j$ :  $receive(row_i^{(l)}, p_{j'})$  indicates that processor  $p_j$  receives contents of  $row_i^{(l)}$  from processor  $p_{j'}$ .
- neighbour(j, k) indicates the neighbour of processor  $p_j$  along the kth dimension of hypercube. If j' = neighbour(j, k) then j' is obtained by complementing the kth bit in the binary representation of j.

• Let d be the dimension of the hypercube and  $l \in \{1, ..., d\}$  be the dimension across which the hypercube is to be divided into two halves. We define two sets  $P_{upper}^{(l)}$  and  $P_{lower}^{(l)}$  as

$$P_{unner}^{(l)} = \{j \mid j > neighbour(j, l)\}$$

$$P_{lower}^{(l)} = \{j \mid j < neighbour(j, l)\}$$

where  $j \in \{0, \dots, p-1\}$ . Further,

$$P_{upper}^{(0)} = \{p/2, p/2 + 1, \dots, p - 1\}$$

$$P_{lower}^{(0)} = \{0, 1, \dots, p/2 - 1\}.$$

In the next two sub-sections the following assumptions hold.

- Each processor contains sufficient local memory and no global memory exists.
- $N/p \ge 1$ , where N is the number of rows of blocks in the block tridiagonal linear system.
- All links between the processors of the hypercube are capable of full-duplex communication.
- For each communication step between a pair of neighbouring processors, the startup time is assumed to be negligible.
- Each processor can overlap its computation with the data communication from/to its neighbours.
- Inversion of matrix blocks is done using the exchange method.
- The matrix blocks  $d_i^{(l)}$ ,  $i=1,\ldots,N$ , are non-singular at all stages  $l=1,\ldots,\log N$ .

## 4.3 Our Improved Mapping of CE onto Hypercubes

Initially, all  $row_i^{(0)}$ ,  $i=1,\ldots,N$  in the block tridiagonal linear system are partitioned into p/2 sets  $S_1^{(0)}, S_2^{(0)}, \ldots, S_{p/2}^{(0)}$  of 2N/p rows each such that

$$S_i^{(0)} = \{row_{2(i-1)\frac{N}{p}+1}^{(0)}, row_{2(i-1)\frac{N}{p}+2}^{(0)}, \dots, row_{2i\frac{N}{p}}^{(0)}\}$$

$$i = 1, \ldots, p/2.$$

One copy of each set  $S_i^{(0)}$  is stored in a pair of processors  $p_j$  and  $p_{j'}$ ,  $j \in \{0, \dots, p/2-1\}$ and  $j' \in \{p/2, \dots, p-1\}$  such that

$$j = E_{\log p - 1}(i - 1)$$

i.e., j=(i-1)th code word of the binary reflected gray code with  $\log p-1$  bits and

$$j' = neighbour(j, \log p)$$

At any stage l of the algorithm, we maintain sets  $C_j^{(l)}$  at every processor  $p_j$  such that

$$C_i^{(l)} = \{row_i^{(l-1)} \mid row_i^{(l)} \text{ is computed at processor } p_j\}.$$

For all  $j \in P_{lower}^{(0)}$ , let  $k = D_{log p-1}(j) + 1$ . Thus the members of the set  $S_k^{(0)}$  are stored at processor  $p_i$ . Initially, let

$$C_i^{(1)} = \{row_i^{(0)} \mid row_i^{(0)} \in S_k^{(0)} \text{ and } i \in \{1, 3, \dots, N-1\}\}$$

i.e.,  $Compute\ row_i^{(l)}$  step is executed at  $p_j$  for all odd indexed equations which are members of the set  $S_k^{(0)}$ . Similarly, for all  $j' \in P_{upper}^{(0)}$ , let  $k = D_{\log p - 1}(neighbour(j', \log p)) + 1$ . Then

$$C_{i'}^{(1)} = \{row_i^{(0)} \mid row_i^{(0)} \in S_k^{(0)} \text{ and } i \in \{2, 4, \dots, N\}\}$$

i.e.,  $Compute\ row_i^{(l)}$  step is executed at  $p_j$  for all even indexed equations which are members of the set  $S_k^{(0)}$ . We now formally present our CE algorithm for hypercubes.

#### Algorithm 4

- (\* Cyclic elimination on hypercube \*)
- 1. for  $j \in \{0, 1, ..., p/2 1\}$  do in parallel
- $p_j, p_{j+p/2}: k = D_{\log p 1}(j) + 1$  $h = 2^{l-1}$ 2.
- 3.
- 4. endfor
- 5. **for** l = 1 **to**  $\log p 1$  **do**
- (\* Elimination phase \*)
- for all  $j \in \{0, \dots, p-1\}$  do in parallel 7.
- $p_j$  : for all i such that (  $row_i^{(l)} \in C_j^{(l)}$  ) do  $Compute \ row_i^{(l)}$ 8.
- 9.
- 10. endfor
- 11. endfor

```
12.
         if (l < \log p - 1) then
               (* Copying phase *)
13.
              for j \in \{0, \dots, p-1\} do in parallel
14.
                          p_i: S_k^{(l)} = C_i^{(l)}
15.
                                for all i such that (row_i^{(l-1)} \in C_i^{(l)})
16.
                                     send(row_i^{(l)}, p_{neighbour(j,l)})
17.
                                     receive(row_{i'}^{(l)}, p_{neighbour(j,l)})
18.
                                     S_{k}^{(l)} = S_{k}^{(l)} \cup \{row_{i}^{(l)}\}
19.
20.
                                endfor
21.
               endfor
              (* Updating C_j^{(l+1)} *)
22.
              for j \in P_{lower}^{(l)} and j' \in P_{upper}^{(l)} do in parallel
23.
                          p_j: min = minimum\{i \mid row_i^{(l)} \in S_k^{(l)}\}
24.
                                C_i^{(l+1)} = \phi
25.
                                for i=min to min+(\frac{N}{p}-1)h step 2h do C_j^{(l+1)}=C_j^{l+1}\cup\{row_i^{(l)}\}
26.
27.
                                endfor
28.
                          p_{j'}: min = minimum\{i \mid row_i^{(l)} \in S_K^{(l)}\}
29.
                                C_{i'}^{(l+1)} = \phi
30.
                                for i = min + h to min + Nh/p step 2h do
31.
                                     C_{i'}^{(l+1)} = C_{j'}^{l+1} \cup \{row_i^{(l)}\}
32.
33.
                                endfor
34.
               endfor
35.
         endif
36. endfor
37. (* Obtaining x_i *)
38. for j \in \{0, \dots, p-1\} do in parallel
          p_j: if (\frac{N}{p}>1) then
39.
                    Solve the independent system of \frac{N}{p} block equations in
40.
                   C_j^{(\log p-1)} using BGE algorithm to obtain
                    \{x_i \mid row_i^{(\log p - 1)} \in C_i^{(\log p - 1)}\}
               else (* N = p *)
41.
                   for all i such that ( row_i^{(\log p-1)} \in C_j^{(\log p-1)} ) do x_i = (d_i^{(\log p-1)})^{-1}v_i^{(\log p-1)}
42.
43.
44.
                    endfor
```

45. endif

46. end

We see that the communication of data occurs in the lines 7-11 (elimination phase) and lines 14-21 (copying phase). Lines 7-11 for computing  $row_i^{(l)}$  at processor  $p_j$  require data of  $row_{i-h}^{(l-1)}$ ,  $row_i^{(l-1)}$ , and  $row_{i+h}^{(l-1)}$ . Of the three,  $row_i^{(l-1)}$  is available on  $p_j$ . If  $row_{i-h}^{(l-1)}$  and  $row_{i+h}^{(l-1)}$  are not available on  $p_j$ , then they have to be brought in from its neighbouring processors. In lines 14-21 of the copying phase, (see figures 7(c),(e)), at every stage l, exactly  $\frac{N}{p}$  rows of blocks are copied in each direction between every pair of neighbouring processors along dimension l-1 of the hypercube. Again the communication is between neighbouring processors only. Hence the number of hops in any communication step is no more than one at any stage of the algorithm.

Note that after  $\log p - 1$  stages, the above algorithm switches over to BGE algorithm on uniprocessor. This is because after  $\log p - 1$  stages each processor contains an independent set of equations which can be solved without communicating with any other processor. Since on a uniprocessor, the BGE algorithm is the most efficient one, switching over to BGE enhances the performance.

## 4.4 Analytical Performance Studies

We now derive expressions for the execution time of our algorithm and also the CR algorithm.

#### 4.4.1 Our Improved CE Algorithm

The lines 1-4 take  $T_1 = 3$  time units to execute in parallel on p processors. In lines 5-36 the copying phase of every iteration l overlaps with the computation phase of (l+1)th iteration. Thus this step (lines 5-36) takes

$$T_{2} = \max\{\left(\frac{N}{p} - 1\right)e, T_{e}\} + e + (\log p - 2)(\max\{\left(\frac{N}{p} - 1\right)e, \frac{N}{p}(T_{e} + 1)\} + T_{e} + e) + (T_{e} + \left(\frac{N}{p} - 1\right)\max(e, T_{e}) + e) \text{ units.}$$

For lines 38-46,  $T_3 = (\frac{N}{p}-1)(e+s)+(2n^3-n^2)$ . Thus the total time taken,  $T_{total} = T_1+T_2+T_3$ . Let us look at the communication complexity of our algorithm without considering any overlap between the communication and computation steps. The contribution from elimination

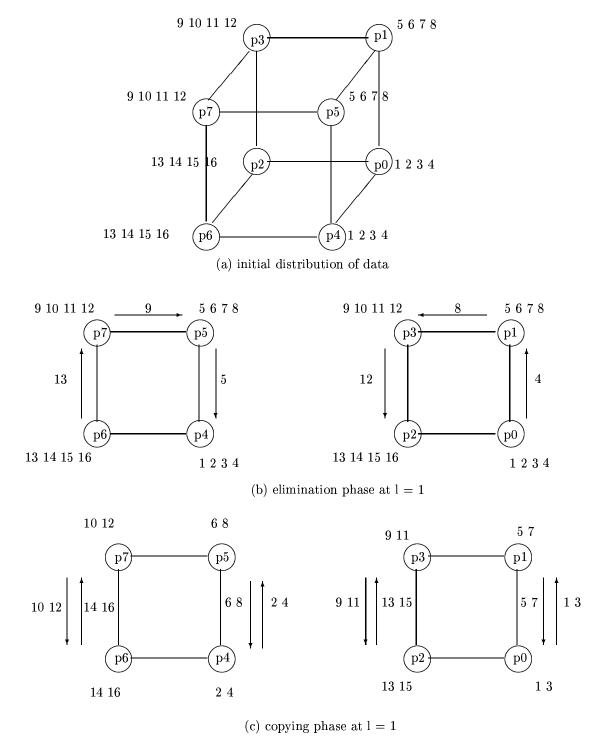


Figure 7: (a) Progression of our algorithm on hypercube for N=16 and p=8

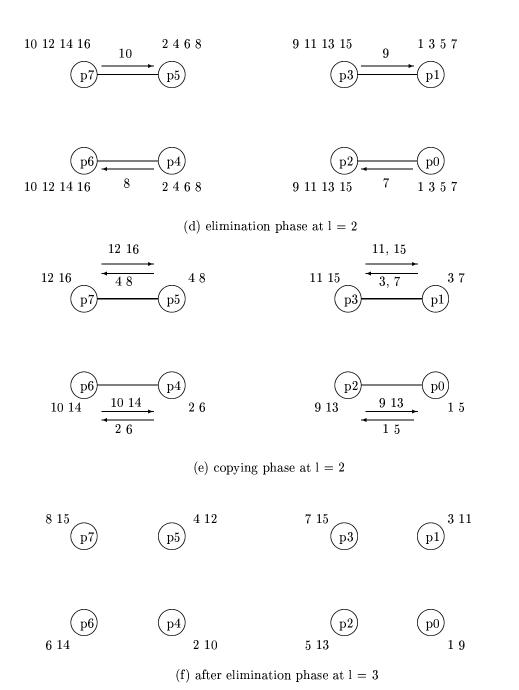


Figure 7: (b) Progression of our algorithm on hypercube for N=16 and p=8

phase (lines 7-11) alone is  $(\log p - 1)T_e$  and that from copying phase (lines 14-21) alone is  $\frac{N}{p}(\log p - 2)T_e$ . Thus the total communication complexity of our algorithm is a sum of these two, given by

$$\left(\left(\frac{N}{p}+1\right)\log p-2\frac{N}{p}-1\right)T_e$$
 units

where  $T_e = 5n^2(C/E)$ ,  $n \times n$  being the size of each block.

#### 4.4.2 CR Algorithm

In reduction phase, the first  $\log(N/p)$  stages involve one hop communication of rows of blocks and  $N/(p2^l)$  computations of  $row_i^{(l)}$  (figure 4(b) and (c)). Here the communication of a row of blocks and  $(\log(N/p) - 1)$  computations of  $row_i^{(l)}$  are overlapped. The  $\log(N/p) + 1$ th stage involves one hop communication of a row of blocks and one  $row_i^{(l)}$  computation step in a non-overlapped manner (figure 4(d)). The remaining  $(\log p - 1)$  stages involve two hop communication of a row of blocks and one  $row_i^{(l)}$  computation step in a non-overlapped manner (figure 4(e)). Thus the total time for the reduction phase works out to be

$$T_{reduction} = (e+4)\log(\frac{N}{p}) + \sum_{l=1}^{\log(\frac{N}{p})} (max\{(N/(p2^{l})-1)(e+1), T_e\} + (T_e+e+4) + (\log p-1)(2T_e+e+4) \text{ units}$$

Similar communication pattern exists for back substitution but in reversed manner. Thus the time taken for back substitution phase works out to be

$$T_{back \ substitution} = (s+3) \log(\frac{N}{p}) + \sum_{l=1}^{\log(\frac{N}{p})} max\{(N/(p2^{l})-1)s, T_{b}\} + (T_{b}+s+3) + (\log p - 1)(2T_{b}+s+3) \text{ units.}$$

Taking a block multiplication step between these two phases into account, the total time  $T_{CR} = T_{reduction} + T_{mult} + T_{backsubstitution}$ . Let us look at the communication complexity of CR algorithm without considering any overlap of the communication and computation steps. The contribution from reduction phase alone is  $(\log N + \log p - 1)T_e$  and contribution from back substitution phase alone is  $(\log N + \log p - 1)T_b$ . Thus the total communication complexity of CR algorithm, as a sum of these two, is given by

$$(\log N + \log p - 1)(T_e + T_b)$$
 units

where  $T_e = 5n^2(C/E)$  and  $T_b = n^2(C/E)$ ,  $n \times n$  being the size of each block.

## 5 Experimental Results

To evaluate the accuracy of the above analytical expressions, we implemented a hypercube simulator in C language and compared the *speedups* obtained from our new mapping of CE algorithm with those obtained from the existing mapping of CR algorithm. We used SPARC Classic machines to carry out our simulations. The parameters that were varied were the number of rows of blocks N(512 and 1024), the block size n(1,2, and 4), the ratio of communication step to computation step C/E (10, 25, 50, and 100), and the number of processors p(1 to 1024). The figures 8- 13 show the comparison of measured speedups of the two algorithms for various values of the above parameters. We observe the following facts.

- For N = p our improved mapping scheme for CE algorithm always gives higher speedup than the CR algorithm.
- Increasing the block size n increases the magnitude of speedups obtained by the two schemes (see figures 11(a),12(a), and 13(a)). Increasing the number of rows of blocks, N, shows up a similar trend (see figures 8 and 11, 9 and 12 and, 10 and 13). On the other hand, as the C/E ratio increases, the magnitude of speedup reduces in both the algorithms (see figures 8(a), (b), and (c)).
- The speedup of CR algorithm tends to saturate and even fall as the number of processors increases. Such a saturation effect is absent from our algorithm in which the speedup progressively increases with the number of processors and reaches its maximum value at N = p.
- The results obtained from the simulation studies compared well with the theoretical predictions obtained from the analytical method. The small differences between the speedups obtained from both the methods arise due to the following reason. The analytical method tries to estimate, as closely as possible, the amount of overlap between the computation and communication steps. However, the exact amount of overlap depends on various factors such as the C/E ratio, precedence constraints between various computation and communication tasks, and routing scheme used in the multiprocessor system. The effect of all these factors on the speedup of the algorithms cannot be encapsulated neatly into a single analytical expression.

## 6 Conclusions

We have proposed a new mapping of the CE algorithm onto hypercube multiprocessors for efficiently solving the block tridiagonal linear system of equations. This mapping maintains the same degree of parallelism throughout and uses the concept of data replication to achieve only neighbouring processor communication at all stages of the processing. We have demonstrated the effectiveness of our mapping by comparing it with the existing mapping of CR algorithm onto hypercubes using both analytical and simulation methods. Further work is possible in the direction of controlling the amount of parallelism in our implementation of the CE algorithm [5]. In its present form, our algorithm switches to the sequential BGE algorithm only after  $\log p - 1$  stages when each processor has an independent set of equations which can be solved without communicating with any neighbour. However, switching over to BGE algorithm at an earlier stage (say k) may lead to further improvements in the performance of our algorithm. Determining the optimal value of k is an open problem.

**Acknowledgements** This work is supported by the Indian National Science Academy, and the Department of Science and Technology.

## References

- [1] Ching-Tien Ho and S. Lennart Johnsson, Optimizing tridiagonal solvers for alternating direction methods on boolean cube multiprocessors, SIAM J. Sci. Stat. Comput., Vol 11, No.3, May 1990, pp. 563-592.
- [2] Don Heller, A survey of parallel algorithms in numerical linear algebra, SIAM Review, Vol.20, No.4, Oct. 1978, pp. 740-777.
- [3] R. Hockney, A fast direct solution of Poisson's equation using Fourier analysis, Journal of ACM, Vol. 12, 1965, pp.95-113.
- [4] S. Lakshmivarahan and S. K. Dhall, Analysis and Design of Parallel Algorithms Arithmetic and Matrix Problems, McGraw-Hill Publishing Company, 1990.
- [5] K. N. Balasubramanya Murthy, New Algorithms for Parallel Solution of Linear Equations on Distributed Memory Multiprocessors, Ph.D. Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, India, 1995.

- [6] Wen-Yang Lin and Chuen-Liang Chen, A parallel algorithm for solving tridiagonal linear systems on distributed memory multiprocessors, Intl. Journal of High Speed Computing, Vol. 6, No. 3, 1994, pp. 375-386.
- [7] Juan C. Agui and Javier Jimenez, A binary tree implementation of a parallel distributed tridiagonal solver, Parallel Computing, Vol. 21, No. 2, 1995, pp. 233-241.
- [8] H. S. Stone, *Parallel tridiagonal equation solvers*, ACM Transactions on Mathematical Software, Vol. 1, No. 4, 1975, pp. 289-307.
- [9] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1977.
- [10] P. Amodio, Optimised cyclic reduction for the solution of linear tridiagonal systems on parallel computers, Computers Mathematics and Applications, Vol.26, No.3, 1993, pp.45-53.
- [11] J. M. Ortega and R. G. Voigt, Solution of partial differential equations on vector and parallel computers, SIAM Review, Vol. 27, No. 2, June 1985, pp. 149-240.
- [12] R. P. Pargas, Parallel solution of elliptic partial differential equations on a tree machine, Ph.D. Thesis, University of North Carolina, Chapel Hill, 1982.
- [13] A. H. Sameh and D. J. Kuck, On stable parallel linear system solvers, Journal of ACM, Vol. 25, No. 1, January 1978, pp. 81-91.
- [14] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis, *Introduction to Parallel Computing Design and Analysis of Algorithms*, Benjamin/Cummings Publishing Company Inc., 1994.
- [15] Buzbee, B. L., G. H. Golub and C. W. Nielson, On direct methods for solving Poisson's equations, SIAM Journal on Numerical Analysis, Vol 7, 1970, pp-627-655.
- [16] R. W. Hockney and C. R. Jesshope, Parallel Computers, Adam Hilger Ltd, 1981.
- [17] G. H. Golub, and C. F. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, MD, 1990.
- [18] S. L. Johnsson, Odd-even Cyclic Reduction on Ensemble Architecture and the Solution of Tridiagonal System of Equations, Technical Report DCS-RR339, Department of Computer Science, Yale University, New Haven, CT, 1984.

[19] D. P. Bertsekas and J. N. Tsitsiklis, Parallel and Distributed Computation - Numerical Methods, Prentice-Hall, Engelwood Cliffs, New Jersy, 1989.

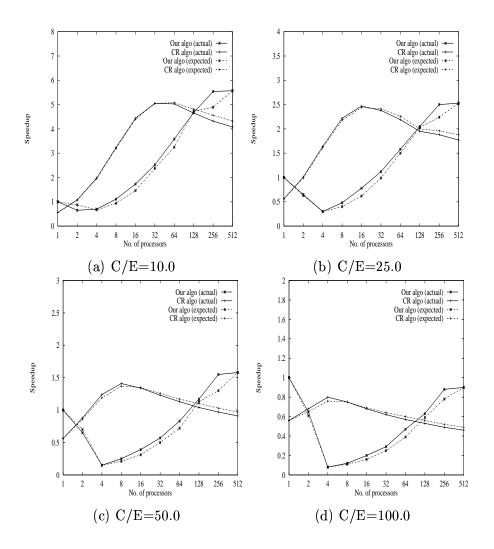


Figure 8: Speedups obtained for our algorithm versus CR algorithm for N=512 and n=1

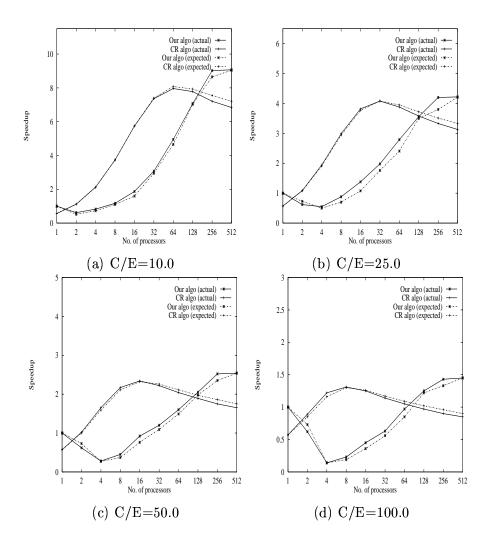


Figure 9: Speedups obtained for our algorithm versus CR algorithm for N=512 and n=2

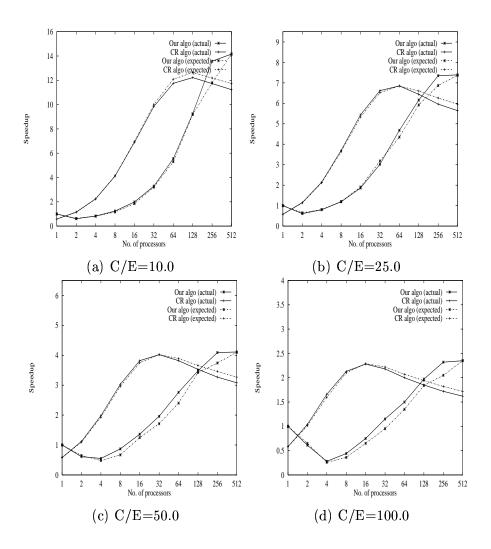


Figure 10: Speedups obtained for our algorithm versus CR algorithm for N=512 and n=4

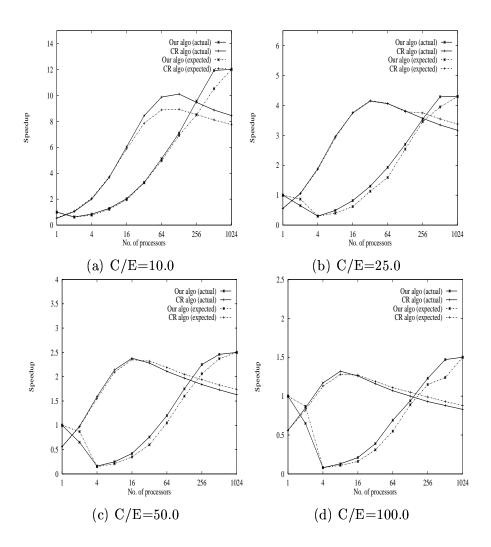


Figure 11: Speedups obtained for our algorithm versus CR algorithm for N=1024 and n=1

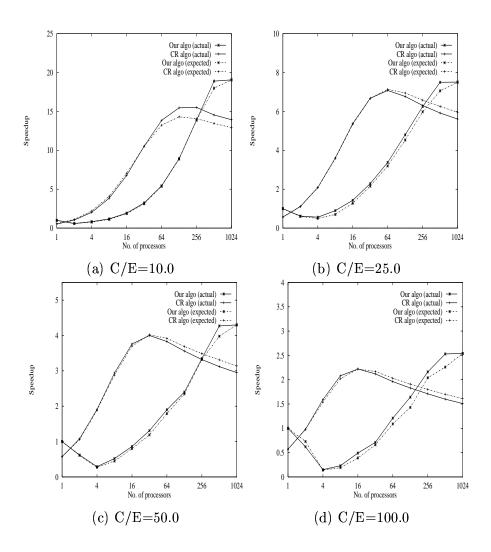


Figure 12: Speedups obtained for our algorithm versus CR algorithm for N=1024 and n=2

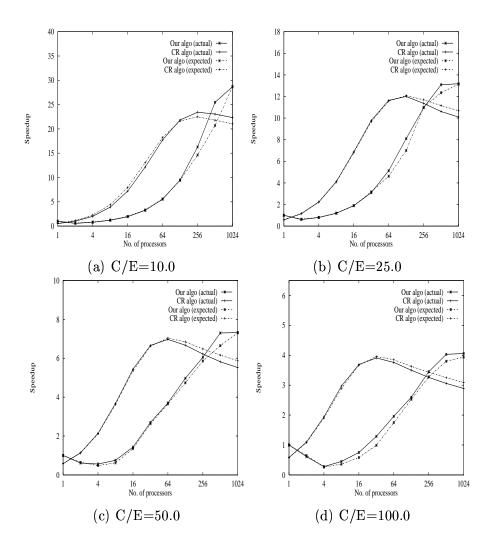


Figure 13: Speedups obtained for our algorithm versus CR algorithm for N=1024 and n=4