Complexity Issues on Designing Tridiagonal Solvers on 2-Dimensional Mesh Interconnection Networks*

Eunice E. Santos

Department of Electrical Engineering and Computer Science
Lehigh University

We consider the problem of designing optimal and efficient algorithms for solving tridiagonal linear systems with multiple right-hand side vectors on two-dimensional mesh interconnection networks. We derive asymptotic upper and lower bounds for these solvers using odd-even cyclic reduction. We present various important lower bounds on execution time for solving these systems including general lower bounds which are independent of initial data assignment, and lower bounds based on classifications of initial data assignments which classify assignments via the proportion of initial data assigned amongst processors. Finally, different algorithms are designed in order to achieve running times that are within a small constant factor of the lower bounds provided.

1. Introduction

In this paper, we consider the problem of designing algorithms for solving tridiagonal linear systems. Such algorithms are referred to as tridiagonal solvers. A method for solving these systems, in which there is particular interest by designers, is the well-known odd-even cyclic reduction method which is a direct method. Due to this interest, we chose to focus our attention on odd-even cyclic reduction. Thus, our results will be applicable to tridiagonal solvers designed on a two-dimensional mesh utilizing cyclic reduction. Much research has been spent exploring this problem which deals with designing and analyzing algorithms that solve these systems on specific types of interconnection networks [1, 5, 7, 8, 9, 10, 11] such as hypercube or butterfly. However, very little has been done on determining lower bounds for solving tridiagonal linear systems [4, 6, 10, 11] on any type of interconnection network or on specific general parallel models [9]. Moreover, few of these papers consider the case of multiple right-hand side (RHS) vectors and the authors know of no papers which derive lower bounds on parallel run-time for odd-even cyclic solvers with multiple RHS vectors.

The main objective of this paper is to present asymptotic upper and lower bounds on the running time for solving tridiagonal systems with multiple right hand sides which utilize odd-even cyclic reduction on 2-dimensional meshes. Our decision to work with meshes is based on the simple fact that they are very common and frequently used interconnection networks. The results obtained in this paper will provide not only a means for measuring efficiency of existing algorithms but also provide a means of pinpointing algorithm design issues, data layouts and communication patterns in order to achieve optimal or near-optimal running times.

While we have already derived results for tridiagonal solvers with only a single vector on a 2-D mesh [10], the results were not easily adaptable to multiple vectors. In fact, multiple vectors produced several layers of complexity towards analysis of this problem. Some of the interesting results we shall show include the following: The skewness in the proportion of data will significantly affect running time. Another interesting result is the proof on the threshold for

* Research partially supported by an NSF CAREER Grant.

processor utilization. For some problem sizes, using common data layouts and straightforward communication patterns do not result in significantly higher complexities than assuming that all processors have access to all data items regardless of communication pattern. In many cases, common data layouts can be used to obtain optimal running times. However, there is no one algorithm, data layout or communication pattern which will provide optimal run-times for all cases. In fact, we present more than 6 algorithms/subalgorithms which are needed in order to show how to achieve optimal or near-optimal running times for all cases.

The paper is divided as follows. Section 2 contains a description of the 2-D mesh topology. In Section 3 we discuss the odd-even cyclic reduction method for solving tridiagonal systems. In Section 4, we derive various important lower bounds on execution. First, we derive general lower bounds for solving tridiagonal systems, i.e. the bounds hold regardless of data assignment. We follow this by deriving lower bounds which rely on categorizing data layouts via the proportion of data assigned amongst processors. Furthermore, we describe commonly-used data layouts designers utilize for this problem. Lastly, we present a variety of algorithms and subalgorithms which will produce running times within a small constant factor of the lower bounds derived. Section 6 gives the conclusion and summary of results.

2. 2-Dimensional Mesh Interconnection Network

A *mesh network* is a parallel model on P processors in which processors are grouped as two types: border processors and interior processors. Each interior processor is linked with exactly four neighbors. Each but for four border processors have three neighbors. And the remaining four border processors have exactly two neighbors. More precisely:

- Denote processors by some $p_{i,j}$ where $1 \le i,j \le \sqrt{P}$
  - For $1 < i,j < \sqrt{P}$ the four neighbors of $p_{i,j}$ are $p_{i+1,j}$, $p_{i-1,j}$, $p_{i,j+1}$, and $p_{i,j-1}$
  - For $i = 1$ and $1 < j < \sqrt{P}$ the three neighbors of $p_{i,j}$ are $p_{i+1,j}$, $p_{i,j+1}$, and $p_{i,j-1}$
  - For $i = \sqrt{P}$ and $1 < j < \sqrt{P}$ the three neighbors of $p_{i,j}$ are $p_{i-1,j}$, $p_{i,j+1}$, and $p_{i,j-1}$
  - For $j = 1$ and $1 < i < \sqrt{P}$ the three neighbors of $p_{i,j}$ are $p_{i+1,j}$, $p_{i-1,j}$, and $p_{i,j+1}$
  - For $j = \sqrt{P}$ and $1 < i < \sqrt{P}$ the three neighbors of $p_{i,j}$ are $p_{i+1,j}$, $p_{i-1,j}$, and $p_{i,j-1}$
  - For $i = 1$ and $j = 1$ the two neighbors of $p_{i,j}$ are $p_{i+1,j}$, and $p_{i,j+1}$
  - For $i = 1$ and $j = \sqrt{P}$ the two neighbors of $p_{i,j}$ are $p_{i+1,j}$, and $p_{i,j-1}$
  - For $i = \sqrt{P}$ and $j = 1$ the two neighbors of $p_{i,j}$ are $p_{i-1,j}$, and $p_{i,j+1}$
  - For $i = \sqrt{P}$ and $j = \sqrt{P}$ the two neighbors of $p_{i,j}$ are $p_{i-1,j}$, and $p_{i,j-1}$

Communication between neighbor processors require exactly 1 time step. By this, we mean that if a processor transmits a message to its neighbor processor *at the beginning* of time x, its neighbor processor will receive the message *at the beginning* of time $x + 1$. Moreover, we assume that processors can be receiving[transmitting] a message while performing a local operation.

3. Odd-Even Cyclic Reduction Method

The Problem: Given M which is an $N \times N$ matrix, and R vectors $\mathbf{b}_1, \mathbf{b}_2 \cdots \mathbf{b}_\mathbf{R}$ each of size N. For each $s (\le R)$, solve $\mathbf{x}_\mathbf{s}$ where $M\mathbf{x}_\mathbf{s} = \mathbf{b}_\mathbf{s}$.

We assume that $\mathbf{b}_1, \mathbf{b}_2 \cdots \mathbf{b}_\mathbf{R}$ can be stored in one matrix B of size $N \times R$ where the s^{th} column of B represents vector $\mathbf{b}_\mathbf{s}$. We make an analogous assumption for $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_\mathbf{R}$. Moreover, we assume for the sake of simplicity that $1 < P = 2^{2k} \le NR$ where k is an integer and $R = 2^r$. An *algorithm* is simply a set of arithmetic operations such that each processor is assigned a sequential list of these operations. An initial assignment of data to the processors is called a *data layout*. A list of message transmissions and receptions between processors is called a *communication pattern*. These three

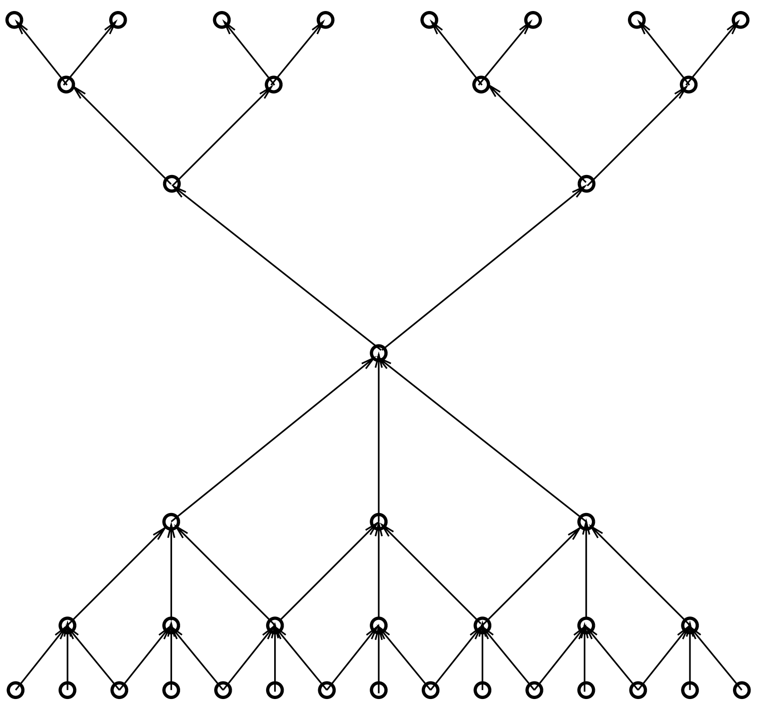

Fig. 1. Row dependency for odd-even cyclic reduction for $N = 15$ and $R = 1$.

components (algorithm, data layout, communication pattern) are needed in order to determine running time.

Odd-even cyclic reduction [2, 3, 7] is a recursive method for solving tridiagonal systems of size $N = 2^n - 1$. This method is divided into two parts: reduction and back substitution.

The first step of reduction is to remove each odd-indexed x_{s_i} and create a tridiagonal system of size $2^{n-1} - 1$. We then do the same to this new system and continue in the same manner until we are left with a system of size 1. This requires n phases. We refer to the tridiagonal matrix of phase j as M^j and the vector as $\mathbf{b}_\mathbf{s}^\mathbf{j}$. The original M and $\mathbf{b_s}$ are denoted M^0 and $\mathbf{b}_\mathbf{s}^0$. The three non-zero items of each row i in M^j are denoted l_i^j, m_i^j, r_i^j (left, middle, right). Below is the list of operations needed to determine the items of row i in matrix M^j (which we refer to as M_i^j.

$$e_i^j = -\frac{l_i^{j-1}}{m_{i-2^{j-1}}^{j-1}}, f_i^j = -\frac{r_i^{j-1}}{m_{i+2^{j-1}}^{j-1}},$$
$$l_i^j = e_i^j l_{i-2^{j-1}}^{j-1}, \quad r_i^j = f_i^j r_{i+2^{j-1}}^{j-1},$$
$$m_i^j = m_i^{j-1} + e_i^j r_{i-2^{j-1}}^{j-1} + f_i^j l_{i+2^{j-1}}^{j-1},$$
$$b_{s,i}^j = b_{s,i}^{j-1} + e_i^j b_{s,i-2^{j-1}}^{j-1} + f_i^j b_{s,i+2^{j-1}}^{j-1}$$

The results in this paper assume that determining values for each e_i^j, f_i^j, l_i^j, r_i^j, m_i^j, or $b_{s,i}^j$ are atomic, i.e. if a processor is computing, for example, e_i^j, then the processor must compute all the operations to satisfy the equation for e_i^j given above.

Clearly, each system is dependent on the previous systems. The lower half of Figure 1 shows the dependency between all the M^j's in the reduction phase. The nodes in level j of the figure represent the rows of M^j. A directed path from node a in level k to node b in level j implies that row a in M^k is needed in order to determine the entries for row b in M^j. In fact, Figure 1 also pinpoints the dependencies between all the $b_{s,i}^j$'s. Since there are no dependencies between some $b_{s_1,i}^j$ and some $b_{s_2,z}^y$ (where $s_1 \neq s_2$), the only dependencies remaining are those between M^j's and $\mathbf{b}_\mathbf{s}^\mathbf{y}$'s. From the definition of the problem, we see that for each s, $b_{s,i}^j$ is dependent on M_i^j.

In this paper, for simplicity, we assume that if an algorithm employs odd-even cyclic reduction, we assume that a processor computed items of whole rows of a matrix (i.e. the three non-zero data items).

The back substitution phase is initiated after the system of one equation has been determined. We recursively determine the values of the x_{s_i}'s. The first operation is $x_{s_{2^{n-1}}} = \frac{b^{n-1}_{s,2^{n-1}}}{m^{n-1}_{2^{n-1}}}$. For the remaining variables x_{s_i}, let j denote the last phase in the reduction step before x_{s_i} is removed, then

$$x_{s_i} = \frac{b^{j-1}_{s,i} - l^{j-1}_i x_{s_{i-2^{j-1}}} - r^{j-1}_i x_{s_{i+2^{j-1}}}}{m^{j-1}_i}.$$

Again, we are assuming the operations to compute the x_{s_i}'s are atomic for $R = 1$. However, for the case of $R > 1$, we divide the operations for x_{s_i} into the following atomic operations:

$$temp_i = \frac{l^{j-1}_i x_{s_{i-2^{j-1}}} + r^{j-1}_i x_{s_{i+2^{j-1}}}}{m^{j-1}_i},$$

and

$$x_{s_i} = \frac{b^{j-1}_{s,i}}{m^{j-1}_i} - temp_i.$$

Returning back to Figure 1, we see that the back substitution is represented by the top half of the graph. In essence, the entire tridiagonal system's dependency graph is $R + 1$ copies of the graph in Figure 1 (one for the M matrix, and the remaining for the R vectors) with some additional edges. Clearly, for any two vectors of **b**, their respective dependency graphs will not have any edges between them. In fact the only edges to be added are from the sub-dependency graph of M to every vector of **b**. In other words, for all vectors $\mathbf{b_s}$, each node in the M sub-dependency graph will have an edge to the "mirror" node in the sub-dependency graph of $\mathbf{b_s}$.

The serial complexity of this method is denoted by $S(N, R, P)$ where $P = 1$. $R = 1$ is a special case, i.e.

$$S(N, 1, 1) = 19N - 14\log(N + 1) - 4.$$

For $R > 1$,

$$S(N, R, 1) = 15N - 10n - 5 + M(6N - 4n - 1).$$

In Section 4, we derive several important lower bounds on running time for odd-even cyclic reduction algorithms on 2-D mesh networks. Specifically, we derive general lower bounds independent of data layout, lower bounds which are based on categorization of data layouts, and lower bounds on running time for common data layouts for this problem. In Section 5.6 we present optimal algorithm running times on 2-d mesh topologies.

4. Lower bounds for Odd-Even Cyclic Reduction

Definition 4.1. *The class of all communication patterns is denoted by $\mathcal{C}$. The class of all data layouts is denoted by $\mathcal{D}$. A data layout D is said to be a single-item layout if each non-zero matrix-item is initially assigned to a unique processor.*

In the following sections we shall provide lower bounds based on certain types of data layouts as well as general lower bounds. The lower bounds hold regardless of the choice of communication pattern.

4.1. A General Lower Bound for Odd-Even Cyclic Reduction

In this subsection we assume the data layout is the one in which each processor has a copy of every non-zero entry of $\underline{M}^0$ and $\mathbf{b}^0_\mathbf{s}$. We denote this layout by D. Since D is the best data layout possible, a lower bound based on this data layout will provide a general lower bound.

Remark 4.1. *We will denote the (parallel) computation lower bound for odd-even cyclic reduction of size N and R and utilizing P processors by $S(N, R, P)$.*

Remark 4.2. *Let A be an odd-even cyclic reduction algorithm. A computation lower bound for A, given P processors and assuming none of the P processors are totally idle is $S(N, R, P) = \Omega(\frac{NR}{P} + \log N)$.*

Theorem 4.1. *Let A be an odd-even cyclic reduction algorithm. The following is a lower bound for A regardless of data layout and communication pattern:*

$$\begin{cases} \max(S(N,R,P), \sqrt{(P/R)}) = \Omega(\frac{NR}{P} + \log N \\ \qquad + \sqrt{(P/R)}), \quad \text{if } P \le RN^{\frac{2}{3}} \\ \max(S(N,R,RN^{\frac{2}{3}})) = \Omega(N^{\frac{1}{3}}), \quad \text{otherwise} \end{cases}$$

Proof. We will begin by assuming that all P processors will be used in computation and/or communication. Clearly, $S(N,R,P)$ is a lower bound for the problem. Furthermore, we see that there are groups of processors which work together in order to solve the items in the dependency graph of M or in any of the $\mathbf{b_s}$'s graphs. Denote these (not necessarily disjoint) groups of processors by $\mathcal{M}_1, \mathcal{M}_2, \cdots \mathcal{M}_z$. Let $P' = \max_{i \le z} |\mathcal{M}_i|$ ($\le \min(N,P)$). Since these processors must ultimately produce only one row of M in the back substitution phase, it is clear there is some type of "all-to-one broadcast" of P' processors in order to complete the back-substitution phase of M and/or $\mathbf{b_s}$. Thus, on a two-dimensional mesh, this will require communication of at least $\sqrt{P'}$. Moreover, this will also require computation of at least $\frac{N}{P'}$.

Therefore, a lower bound, assuming all P processors are used, is $\max(S(N,R,P), \frac{N}{P'}, \sqrt{P'})$. This leads to the fact that $P' = \max(1, \frac{P}{R})$ in order to minimize the lower bound. Furthermore, we see that using more than $\Omega(RN^{\frac{2}{3}})$ processors, will reduce efficiency; i.e. the threshold for procesor utilization is $\Omega(RN^{\frac{2}{3}})$. □

In Section 5.6 we will provide optimal algorithms, i.e. the running times are within a small constant factor of the lower bounds. Therefore, we note that our results show that when P is sufficiently large, i.e. $P = \Omega(RN^{\frac{2}{3}})$ using more than $RN^{\frac{2}{3}}$ processors will not lead to any substantial improvements in running time if utilizing data layout D.

4.2. Lower Bounds for $\mathcal{O}$ on $\frac{c}{P}$-data layouts

Many algorithms designed for solving tridiagonal linear systems assume that the data layout is single-item and that each processor is assigned roughly $\frac{1}{P}^{th}$ of the rows of M^0 and the items of b where P is the number of processors available. However, in order to determine whether skewness of proportion of data has an effect on running time and if so, exactly how much, we classify data layouts by initial assignment proportions to each processor. In other words, in this section, we consider single-item data layouts in which each processor is assigned at most a fraction $\frac{c}{P}$ of the rows of M and items of b where $1 \le c \le P(\frac{NR+N-P+1}{NR+N})$.

Definition 4.2. *Consider c where $1 \le c < P(\dfrac{NR+N-P+1}{NR+N})$. A data layout D on P processors is said to be a $\frac{c}{P}$-data layout if*

(a) D is single-item,

(b) no processor is assigned more than a fraction $\frac{c}{P}$ of the rows of M^0 or items of b,

(c) at least one processor is assigned exactly a fraction $\frac{c}{P}$ of the rows of M^0 or items of b, and

(d) each processor is assigned at least one row of M^0 or items of b.

Denote the class of $\frac{c}{P}$ data layouts by $\mathcal{D}(\frac{c}{P})$.

Theorem 4.2. *If $D \in \mathcal{D}(\frac{c}{P})$, then for any $A \in \mathcal{O}$ for R vectors, a lower bound is*

$$\max(S(N,R,P), \frac{\sqrt{P}}{8}, \frac{NRc}{6P}) \\ = \Omega(\frac{NR}{P} + \log N + \sqrt{P} + \frac{NRc}{P})$$

Proof. The computational lower bound must also be a lower bound for the problem regardless of communication. Thus $S(N,R,P)$ is a lower bound. Moreover, since at least one processor is assigned $\frac{NRc}{P}$ pieces of data, this processor (denoted as p_0) can either choose to use that data item in a binary arithmetic operation or send out the item to another processor. Therefore, it is clear that $\frac{NRc}{6P}$ must also be a lower bound.

We will now describe the argument for why $\frac{\sqrt{P}}{8}$ must also be a lower bound. We will use a contradiction argument.

Consider any two data items from M and/or B. These two items are said to be "related" if they are both required by some computation in order to provide final solutions. Clearly, any initial data item of M, i.e. M^0 will be related to any other initial item of M. Suppose the $\frac{\sqrt{P}}{8}$ is not a lower bound, this implies that any two processors which are both assigned initial items of M cannot be more than or equal to a distance of $\frac{\sqrt{P}}{4}$ from each other on the 2-D mesh. Furthermore, consider a processor p which computes M^{n-1}. Processor p will require data from all of the items of M^0. Clearly, only processors which are within at most a distance of $\frac{\sqrt{P}}{8} - 1$ can be assigned items from M^0. Furthermore, expanding the distance from $\frac{\sqrt{P}}{8} - 1$ to $\frac{\sqrt{P}}{4} - 2$ from p will now contain all the processors which will utilize any item from M^0 for computation. Finally expanding the distance from p to $\frac{3\sqrt{P}}{8} - 3$ will represent all the processors that are assigned data items from not only M^0 but also items of B^0 which are related to any item in M^0. Every processor is assigned at least one initial piece of data and the layout is single-item. Moreover, every item in B^0 is related to at least one item of M^0. Thus, every processor assigned an item of B^0 or M^0 must be inside this region. However, the region does not include all the processors in the mesh. Therefore, this is a contradiction. □

Analyzing the result given in the above theorem, we see that $\frac{1}{P}$-data layouts will result in the best lower bounds for this class of data layouts, i.e. :

Corollary 4.1. *If $D \in \mathcal{D}(\frac{1}{P})$, then for any $A \in \mathcal{O}$ for R vectors, a lower bound is*

$$\max(S(N,R,P), \sqrt{P}) = \Omega(\frac{NR}{P} + \log N + \sqrt{P})$$

Comparing results against the general lower bound, we see that for sufficiently large N (i.e. $N >> P$, or more precisely $P \leq R(N)^{\frac{2}{3}}$), the lower bound for $\frac{1}{P}$-data layouts is precisely equal to the general lower bound. The complexity of any algorithm $A \in \mathcal{O}$ using a $\frac{1}{P}$-data layout is $\Omega(\log N + \frac{NR}{P} + \sqrt{P})$. In Section 5.6, we present algorithms that are within a small constant factor of the bounds presented in this and previous subsections.

5. Optimal Algorithms, Data Layouts, and Communication Schedules

In this section, we design the algorithms which will produce running times which are within a small constant factor of the lower bounds derived in the last section. We begin by discussing the data layouts we plan to use. We follow this with a discussion of some of the communication schedules (mostly dealing with distribution/broadcast of the data to processors that require this information for computation). Lastly, we present the optimal algorithms along with their running times. We will discuss under which circumstances one algorithm should be used over another in order to achieve optimal or near-optimal running times.

5.1. Definitions of Data Layouts

In the previous sections, we have derived various lower bounds on running time for tridiagonal solvers which utilize odd-even cyclic reduction on a 2-dimensional mesh interconnection topology. In order to design algorithms which are within a constant factor of these lower bounds, we must determine which types of data layouts we will use. In this section, we present definitions on data layouts in order to use these layouts in the next subsections in our algorithm designs.

We would like to point out that_while we can easily use the best data layout D to obtain the general lower bounds, in this section we will present data layouts which are not so powerful but which still achieve the lower bounds stated. In fact, in the cases where $N >> P$, no processor is assigned more than $O(\frac{NR}{P})$ data items.

We begin by presenting partial layouts and in each subsection, we discuss a data layout we will utilize in our optimal algorithms.

Definition 5.1. *Let P be an integer where $1 \leq P \leq N$ and $2^i - 1 \leq P < 2^{i+1} - 1$. A data layout on P processors, $p_1, \cdots p_P$, is an M-replicative-blocked data layout if for all $j < 2^{i-1}$, p_j is assigned the nonzero items in rows $(j-1)2^{n-i}+1$ to $(j+1)2^{n-i} - 1$ of M^0. We denote this layout by $D_{M,P}$.*

Definition 5.2. *Let P be an integer where $1 \leq P \leq N$ and $2^i - 1 \leq P < 2^{i+1} - 1$. A data layout on P processors, $p_1, \cdots p_P$, is an $\mathbf{b_s}$-replicative-blocked data layout if for all $j < 2^{i-1}$, p_j is assigned $b_{s,l}$ where $(j-1)2^{n-i} + 1 \leq l \leq (j+1)2^{n-i} - 1$. We denote this layout by $D_{\mathbf{b_s},P}$.*

Definition 5.3. *Let P be an integer where $1 \leq P \leq N$ and $P = 2^i$. A data layout on P processors, $p_1, \cdots p_P$, is an M-single-item-blocked data layout if (a) for all $j < P-1$, p_j is assigned the nonzero items in rows $(j-1)\frac{N}{P-1} + 1$ to $j\frac{N}{P-1}$ of M^0, (b) p_{P-1} is assigned the nonzero items in rows $(P-2)\frac{N}{P-1} + 1$ to $(P-1)\frac{N}{P-1} - 1$ of M^0, and (c) p_P is assigned the nonzero items in row N of M^0. We denote this layout by $D'_{M,P}$.*

Definition 5.4. *Let P be an integer where $1 \leq P \leq N$ and $2^i - 1 \leq P < 2^{i+1} - 1$. A data layout on P processors, $p_1, \cdots p_P$, is an $\mathbf{b_s}$-single-item-blocked data layout (a) for all $j < P-1$, p_j is assigned the vector items $(j-1)\frac{N}{P-1} + 1$ to $j\frac{N}{P-1}$ of $\mathbf{b_s}$, (b) p_{P-1} is assigned the nonzero items in vector items $(P-2)\frac{N}{P-1} + 1$ to $(P-1)\frac{N}{P-1} - 1$ of $\mathbf{b_s}$, and (c) p_P is assigned the vector item N of $\mathbf{b_s}p$. We denote this layout by $D'_{\mathbf{b_s},P}$.*

5.1.1. Data Layout 1

If $P \geq RN^{\frac{2}{3}}$ then we will only use $R2^{2\lfloor\frac{n-1}{3}\rfloor}$ processors, and, in this case, we will simply assume $P = R2^{2\lfloor\frac{n-1}{3}\rfloor}$.

This data layout will be utilized in order to achieve the general lower bounds when $R \leq P$.

We begin by partitioning the processors into R groups of $\frac{P}{R} = 2^{2k-r}$ processors. For the sake of simplicity, we will assume that r is even. These R groups will be referred to as $\mathcal{P}_{i,j}$ where $1 \leq i, j \leq \sqrt{R}$.

The processors assigned to group $\mathcal{P}_{s,t}$ are $p_{i,j}$ where $(s-1)2^{k-\frac{r}{2}} + 1 \leq i \leq s2^{k-\frac{r}{2}}$ and $(t-1)2^{k-\frac{r}{2}} + 1 \leq j \leq t2^{k-\frac{r}{2}}$.

We will create a new notation for the mesh processors in order to utilize the partial data layouts defined in this section. Consider the processors in group $\mathcal{P}_{s,t}$, these processors will be denoted by $p_1^{s,t}, p_2^{s,t}, \cdots p_{\frac{P}{R}}^{s,t}$ where $p^{(s-1)2^{k-\frac{r}{2}}+i,(t-1)2^{k-\frac{r}{2}}+j}$ is denoted by

- $p^{s,t}_{(i-1)\sqrt{P/R}+j}$ if i is odd
- $p^{s,t}_{(i-1)\sqrt{P/R}+\sqrt{P/R}-j+1}$ if i is even

For each group $\mathcal{P}_{s,t}$, we will allocate the items of M using data layout $D_{M,\frac{P}{R}}$. Moreover, this group of processors will be allocated vector $\mathbf{b}_{\mathbf{(s-1)\frac{P}{R}+t}}$ using data layout $D_{\mathbf{b}_{\mathbf{(s-1)\frac{P}{r}}},\frac{P}{R}}$.

5.1.2. Data Layout 2

This data layout will be utilized in order to achieve the general lower bounds when $R > P$.

We begin by partitioning the assignment of the $\mathbf{b_s}$ vectors to the P processors. Each processor will be assigned $\frac{R}{P}$ whole vectors. In particular $p_{i,j}$ will be assigned the items in vectors $\mathbf{b_l}$ where $(i-1)\frac{R}{\sqrt{P}} + (j-1)\frac{R}{P} + 1 \leq l \leq (i-1)\frac{R}{\sqrt{P}} + (j)\frac{R}{P}$. Moreover, every processor will be assigned the items in matrix M.

5.1.3. Data Layout 3

This is a single-item data layout and will be utilized in order to achieve the single-item data layout lower bounds when $P < N$.

We will create a new notation for the mesh processors in order to utilize the partial data layouts defined in this section. The processors will be denoted by $p_1, p_2, \cdots p_P$ where $p^{i,j}$ is denoted by

- $p_{(i-1)\sqrt{P}+j}$ if i is odd
- $p_{(i-1)\sqrt{P}+\sqrt{P}-j+1}$ if i is even

We will allocate the items of M using data layout $D'_{M,P}$. Moreover, the processors will be allocated each vector $\mathbf{b_s}$ using data layout $D'_{\mathbf{b_s},P}$. Clearly, this is a $\frac{c}{P}$-data layout where c is constant.

5.1.4. Data Layout 4

This is a single-item data layout and will be utilized in order to achieve the single-item data layout lower bounds when $P > N$ and $R \leq P$.

We begin by partitioning the processors into R groups of $\frac{P}{R} = 2^{2k-r}$ processors. These R groups will be referred to as $\mathcal{P}_i$ where $1 \leq i \leq R$.

If $R \leq \sqrt{P}$ then the processors assigned to group $\mathcal{P}_l$ are $p_{i,j}$ where $(l-1)2^{k-r} + 1 \leq i \leq l2^{k-r}$ and $1 \leq j \leq \sqrt{P}$.

We will create a new notation for the mesh processors in order to utilize the partial data layouts defined in this section. Consider the processors in group $\mathcal{P}_l$, these processors will be denoted by $p^l_1, p^l_2, \cdots p^l_{\frac{P}{R}}$ where $p_{(l-1)2^{k-r}+i,j}$ is denoted by

- $p^l_{(i-1)\sqrt{P}+j}$ if i is odd
- $p^l_{(i-1)\sqrt{P}+\sqrt{P}-j+1}$ if i is even

However, if $R > \sqrt{P}$ then the processors assigned to group $\mathcal{P}_l$ where $l = x\frac{R}{\sqrt{P}} + y$ are $p_{i,j}$ where $i = x + 1$ and $(y-1)\frac{P}{R} + 1 \leq j \leq y\frac{P}{R}$.

We will create a new notation for the mesh processors in order to utilize the partial data layouts defined in this section. Consider the processors in group $\mathcal{P}_l$, these processors will be denoted by $p^l_1, p^l_2, \cdots p^l_{\frac{P}{R}}$ where $p^{(l-1)2^{k-r}+i,j}$ is denoted by $p^l_{(i-1)\sqrt{P}+j}$.

Regardless of the size of R, for group $\mathcal{P}_1$, we will allocate the items of M using data layout $D'_{M,\frac{P}{R}}$. Moreover, each group $\mathcal{P}_l$ will be allocated vector $\mathbf{b_l}$ using data layout $D'_{\mathbf{b_l},\frac{P}{R}}$. Clearly, this is a $\frac{c}{P}$-data layout where c is constant.

5.1.5. Data Layout 5

This is a single-item data layout and will be utilized in order to achieve the single-item data layout lower bounds when $R > P > N$.

We begin by partitioning the assignment of the $\mathbf{b_s}$ vectors to the P processors. Each processor will be assigned $\frac{R}{P}$ full vectors. In particular $p_{i,j}$ will be assigned the items in vectors $\mathbf{b_l}$ where $(i-1)\frac{R}{\sqrt{P}} + (j-1)\frac{R}{P} + 1 \leq l \leq (i-1)\frac{R}{\sqrt{P}} + (j)\frac{R}{P}$. Moreover, processor $p_{1,1}$ will be assigned the items in matrix M. Clearly, this is a $\frac{c}{P}$-data layout where c is constant.

The data layouts presented in this subsection will be used to design optimal or near-optimal algorithms for solving tridiagonal systems on a 2-dimensional mesh. We again note that while we could have simply assumed the best data layout for designing our algorithms to achieve the general lower bounds, we instead have listed data layouts in which no processor has been assigned more than $O(\frac{NR}{P})$ initial data items. Furthermore, we have presented several single item data layouts which we will be using in our optimal algorithm designs in later subsections.

In the following subsection, we present some communication schedules which we will use in our algorithms designs. These schedules are used to address data redistribution.

5.2. Communication Schedules

For some of the algorithms we will design in the next subsection, it is important that we are able to redistribute the data from one layout to another. In this section, we present communication schedules which do the needed redistributions.

5.3. Schedule 1

In this communication schedule, we are redistributing data layout 3 into the layout in which the matrix M is distributed using $D_{M,P}$ and each $\mathbf{b_s}$ is distributed using $D_{\mathbf{b_s},P}$. Clearly, only neighbor processors (as ordered in the data layout discussion for 3) need to communicate.

This would be accomplished by running the following:

```
For all $j \leq P - 2$, processors $p_j$ do in parallel
    send all initially assigned data items to processor $p_{j+1}$
For all $j \geq 2$, processor $p_j$ do in parallel
```

```
    send all initially assigned data items to
    processor $p_{j-1}$
```

Running Time: $O(\frac{RN}{P})$.

5.4. Schedule 2

In this communication schedule, we are redistributing data layout 4 into the layout in which the matrix M is distributed to each group using layout $D_{M,\frac{P}{R}}$. Moreover, we also wish to redistribute the b vectors such that instead of $D'_{b,\frac{P}{R}}$, we wish the distribution to be $D_{b,\frac{P}{R}}$. Two slightly different schedules for this redistribution are needed: one for $R \leq \sqrt{P}$ and one for $R > \sqrt{P}$.

We will discuss the schedule for $R \leq \sqrt{P}$. The redistribution schedule for the other case is very similar and, for brevity, we will omit its discussion.

Consider $R \leq \sqrt{P}$. We begin by redistributing so that each group is assigned the matrix M using $D'_{M,\frac{P}{R}}$. Clearly, only processors above and below (on the mesh) a processor need the same data to communicate. Collecting the data items of M in each column to the first processor in the column and then sending out the information to every other processor in the column would effectively accomplish the first step of the redistribution. The remaining redistributions can be easily taken care of using Schedule 1 above.

This would be accomplished by running the following:

```
For all $j \leq \sqrt{P}$ do in parallel
    For all $2 < i \leq \frac{\sqrt{P}}{R}$ every processor $p_{i,j}$ do
    in parallel
        send all initial data items of $M$ to $p_{1,j}$
        (pipelined broadcast)
    $p_{1,j}$ do a one to all (pipelined) broadcast of
    the items of $M$ to processors $p_{2,j}, \cdots, p_{\sqrt{P},j}$.
For each group $\mathcal{P}_l$ do in parallel
    Run schedule 1 for the assigned vector
    Run schedule 1 for $M$
```

Running Time: $O(\frac{RN}{P} + \sqrt{P})$.

5.5. Schedule 3

In this communication schedule, we are redistributing data layout 5 into the layout in which the matrix M is distributed to every processor. This is simply a "one-to-all broadcast" of $3(N-1)$ items. Clearly, this can be accomplished in $O(N-1+\sqrt{P})$ steps. Due to the assumptions made in layout 5, the running tims is $O(\frac{NR}{P} + \sqrt{P} + \log N)$.

5.6. The Optimal Algorithms

In the previous subsections, we have discussed the data layouts and some communication redistribution schedules we plan to use in our algorithms. In this section, we present the algorithms which will produce running times within at most a small constant factor of the lower bounds presented in the previous section.

5.6.1. SubAlgorithm 1 for Best Data Layout

This algorithm solves a tridiagonal system by sequentially solving values based on values it has already computed and not sent from another processor. This Subalgorithm will be a building block for the full algorithms.

Assume y of the vectors of b are assigned to this group of $P' = 2^q$ processors. Moreover, assume the processors are ordered using the ordering given in data layout 3. The items of row i of level j are considered to be l_i^j, r_i^j, m_i^j and $b_{s,i}^j$ (of all y vectors). Lastly, we will utilize only $P'-1$ processors.

SubAlgorithm 1 for **Data Layout** $\bar{D}$

```
Every processor $p_i$ $(1 \leq i < 2^q)$ do in parallel :
    for $s = 1$ to $n - q$ do
        for $z = 2^{n-q-s}(i-1) + 1$ to
        $2^{n-q-s}(i+1) - 1$ do
            compute (serially) the items of row $z$ of
            level $s$
    send items of row $i$ of level $n-q$ to $p_{i+1}$ and
    receive from $p_{i-1}$
    /* if $i = P' - 1$ do not send, and if $i = 1$ do not receive */
  send items of row $i$ of level $n - q$ to $p_{i-1}$ and receive from $p_{i+1}$
    /* if $i = 2^q - 1$ do not receive, and if $i = 1$ do not send */
  for $s = 1$ to $q - 1$ do
    if $\frac{i}{2^s}$ is an integer then
      compute the items of row $2^{-s}i$ of level $n - q + s$
      if $s \leq \frac{1}{2}\log P$ then
        send items of row $i2^{-s}$ of level $n-q+s$ to $p_{i+2^s}$ routed through the path
          $(p_{i+1}, p_{i+2}, \cdots p_{i+k}, p_{i+k+1}, \cdots p_{i+2^s-1})$ and received from $p_{i-2^s}$
          /* if $i + 2s > 2^q - 1$ do not send, and if $i - 2^s < 1$ do not receive */
        send items of row $i2^{-s}$ of level $n-q+s$ to $p_{i-2^s}$ routed through the path
          $(p_{i-1}, p_{i-2}, \cdots p_{i-k}, p_{i-k-1}, \cdots p_{i-2^s-1})$ and received from $p_{i+2^s}$
          /* if $i + 2s > 2^q - 1$ do not receive, and if $i - 2^s < 1$ do not send */
      else $s = \frac{1}{2}\log P + k$
        send items of row $i2^{-s}$ of level $n-q+s$ to $p_{i+2^s}$ routed through the path
          $(p_{i+\sqrt{P}}, p_{i+2\sqrt{P}}, \cdots p_{i+j\sqrt{P}}, p_{i+(j+1)\sqrt{P}}, \cdots p_{i+(2^k-1)\sqrt{P}})$ and received from $p_{i-2^s}$
          /* if $i + 2s > 2^q - 1$ do not send, and if $i - 2^s < 1$ do not receive */
        send items of row $i2^{-s}$ of level $n-q+s$ to $p_{i-2^s}$ routed through the path
          $(p_{i-\sqrt{P}}, p_{i-2\sqrt{P}}, \cdots p_{i-j\sqrt{P}}, p_{i-(j-1)\sqrt{P}}, \cdots p_{i-(2^k-1)\sqrt{P}})$ and received from $p_{i+2^s}$
          /* if $i + 2s > 2^q - 1$ do not receive, and if $i - 2^s < 1$ do not send */
  For back substitution, if $p_i$ was the last processor to handle items of row $j$
    then $p_i$ computes all $x_{s_i}$ where $b_s$ is assigned to $p_i$ (and if necessary,
    send and receive appropriate values of all $x_{s_m}$)
```

Running Time: $O(\frac{Ny}{P'} + y\log P' + \sqrt{P'})$

5.7. Algorithm 1 for Data Layout 1

This data layout is used when $R \leq P$. This algorithm is simply solving the tridiagonal system with one vector. We have partitioned the processors into R groups. Since the best data layout actually is not needed for subalgorithm 1 and in fact our data layout is sufficient, we simply call SubAlgorithm 1 for each group in parallel with $P' = \frac{P}{R}, y = 1$. Therefore the total running time is $O(\frac{NR}{P} + \log\frac{P}{R} + \sqrt{\frac{P}{R}}) = O(\frac{NR}{P} + \log N + \sqrt{\frac{P}{R}})$.

As we stated in the subsection defining Data Layout 1, if $P \geq RN^{\frac{2}{3}}$, we simply assume $P = RN^{\frac{2}{3}}$. Therefore the running time is clearly $O(N^{\frac{1}{3}})$.

5.8. Algorithm 2 for Data Layout 2

This data layout is used when $R > P$. This algorithm is simply sequentially solving the tridiagonal system with $\frac{R}{P}$ right-hand side vectors. We have partitioned the vectors into P groups. Each processor in parallel, serially solves their assigned system. Therefore the total running time is $O(\frac{NR}{P} + \log N) = O(\frac{NR}{P} + \log N + \sqrt{\frac{P}{R}})$.

5.9. Algorithm 3 for Data Layout 3

This algorithm first calls schedule 1, then it simply solves the tridiagonal system with R vectors. We consider the case $P < \frac{N}{\log N}$. Since the best data layout is actually not needed for subalgorithm 1 and in fact our data layout is sufficient (after schedule 1), we simply call SubAlgorithm 1 with $P' = P, y = R$. Therefore the total running time is the time for Schedule 1 plus the time for Subalgorithm 1: $O(\frac{NR}{P} + R\log N + \sqrt{P})$.

Since $P < \frac{N}{\log N}$ then the run-time is clearly $O(\frac{NR}{P} + \sqrt{P} + \log N)$.

For the case where $\frac{N}{\log N} \leq P \leq N$, subalgorithm 1 is easily modified so that the tasks in the last phases of reduction and the first phases of back substitution are replicated so that at least half the processors are always working. This will produce a running time of: $O(\frac{NR}{P} + \log N + \sqrt{P})$.

5.10. Algorithm 4 for Data Layout 4

This algorithm first calls schedule 2 then it is simply solving the tridiagonal system with one vector. We have partitioned the processors into R groups. Since the best data layout is actually not needed for subalgorithm 1 and in fact our data layout is sufficient (after schedule 2), we simply call SubAlgorithm 1 for each group in parallel with $P' = \frac{P}{R}, y = 1$. Therefore the total running time is $O(\frac{NR}{P} + \log N + \sqrt{P})$.

5.11. Algorithm 5 for Data Layout 5

This algorithm first calls schedule 3, then it is simply sequentially solving the tridiagonal system with $\frac{R}{P}$ right-hand side vectors. We have partitioned the vectors into P groups. Each processor in parallel, serially solves their assigned system. Therefore the total running time is $O(\frac{NR}{P} + \log N + \sqrt{P})$.

In this section, we designed algorithms for each of the data layouts which cover all problem sizes. Analyzing the running times of our algorithms, we see that they are all within a small constant factor of the lower bounds provided. In particular, the algorithms for Data Layouts 1 and 2 satisfy the general lower bounds. The algorithms for the remaining Data Layouts all satisfy the single-item bounds. Moreover, analyzing the design of the algorithms, we see that to achieve the single-item bounds simply requires redistribution of a single-item layout to the layouts used to achieve the general lower bounds. Furthermore, no layout assigned any processor more than $O(\frac{NR}{P})$ initial data items.

6. Conclusion

In this paper we examined the problem of designing optimal tridiagonal solvers on 2-dimensional mesh interconnection networks with multiple right hand sides using odd-even cyclic reduction. This is the first paper to have tackled and solved this problem asymptotically.

We were able to derive asymptotic lower bounds on the execution time. Moreover, we were able to provide algorithms whose running times differ by only a small constant factor of these lower bounds.

Specifically, we proved that the complexity for solving tridiagonal linear systems regardless of data layout (i.e. a general parallel lower bound on a 2-dimensional mesh) is $\Omega(N^{\frac{1}{3}})$ when $P = \Omega(RN^{\frac{2}{3}})$ else the bound is $\Omega(\frac{NR}{P} + \log N + \sqrt{\frac{P}{R}})$. This shows that utilizing more than $\Omega(RN^{\frac{2}{3}})$ processors will not result in any substantial runtime improvements. In fact, utilizing more processors may result in much slower run-times.

When we added the realistic assumption that the data layouts are single-item and the number of data items assigned to a processor is bounded, we derived lower bounds for classes of data layouts. The asymptotic lower bound is $\Omega(\frac{NR}{P} + \log N + \sqrt{P})$. We provided three types of data layouts which provided optimal running times. In other words, we showed that the best types of layouts in general are those in which either (a) each processor is assigned an equal number of rows of M^0 and b, (b) the processors are grouped into R groups in which every group is assigned $\frac{P}{R}$ of the vectors, or (c) each processor is assigned $\frac{R}{P}$ whole vectors of b. Therefore, we see that the skewness of proportion of data will significantly affect performance. Comparing the lower bounds for the $\frac{c}{P}$-layouts with the general lower bounds, we see that restricting the proportion of data items assigned to a processor to $\frac{NR}{P}$ does not result in a significantly higher complexity than assuming all processors have all the data items for sufficiently large N (i.e. $P \leq RN^{\frac{2}{3}}$).

Lastly, we show that there are algorithms, data layouts, and communication patterns whose running times are within a constant factor of the lower bounds provided. This provides us with the Θ-bounds stated above.

To achieve the general lower bound, i.e. the complexity for these methods regardless of data layout, while we could have used D, the best data layout, i.e. the data layout in which every processor is assigned all the data items, we did not. We presented two data layouts in which no processor was assigned more than $O(\frac{NR}{P})$ data items. Two types of algorithms produce optimal running times: (a) algorithms in which processors are partitioned into R sets such that each set is in essence an independent tridiagonal system, or (b) algorithms in which each processor sequentially solves $\frac{R}{P}$ of the vectors.

As stated previously, three different data layouts were provided in order to design optimal algorithms for single-item layouts. The three types of algorithms are: (a) algorithms which in essence utilize block layouts and solve the tridiagonal system R times. (b) algorithms in which processors are partitioned into R sets such that each set is in essence an independent tridiagonal system, or (c) algorithms in which each processor sequentially solves $\frac{R}{P}$ of the vectors. For (b)-(c), the algorithms provided are very similar to those to achieve the optimal running times (regardless of data layout). In fact, the difference in run-time is primarily the need to do redistributions of data in order to run the optimal algorithms.

Finally, it is worthwhile to observe that for sufficiently large N, the single-item lower bounds are asymptotic to the general lower bound.

References

[1] C. AMODIO AND N. MASTRONARDI, A parallel version of the cyclic reduction algorithm on a hypercube, *Parallel Computing*, 19, 1993.

[2] D. HELLER, A survey of parallel algorithms in numerical linear algebra, *SIAM J. Numer. Anal.*, 29(4), 1987.

[3] A. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam-Hilger, 1981.

[4] S. L. JOHNSSON, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Stat. Comput.*, 8, 1987.

[5] S. P. KUMAR, Solving tridiagonal systems on the butterfly parallel computer, *International J. Supercomputer Applications*, 3, 1989.

[6] S. LAKSHMIVARAHAN AND S. D. DHALL, A Lower Bound on the Communication Complexity for Solving Linear Tridiagonal Systems on Cube Architectures, In *Hypercubes 1987*, 1987.

[7] S. LAKSHMIVARAHAN AND S. D. DHALL, *Analysis and Design of Parallel Algorithms : Arithmetic and Matrix Problems*, McGraw-Hill, 1990.

[8] F. T. LEIGHTON, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan Kaufmann, 1992.

[9] E. E. SANTOS, Optimal Parallel Algorithms for Solving Tridiagonal Linear Systems, In *Springer-Verlag Lecture Notes in Computer Science #1300 (Proceedings of Euro-Par)*, 1997.

[10] E. E. SANTOS, Optimal Tridiagonal Solvers on Mesh Interconnection Networks In *Springer-Verlag Lecture Notes in Computer Science #1557 (Proceedings of ACPC)*, 1999.

[11] E. E. SANTOS, Solving Tridiagonal Linear Systems on a Torus *Proceedings of the International Conference on Parallel and Distributed Processing and Techniques*, 1998.

Received: May 15, 1999
Accepted in revised form: January 21, 2000

Contact address:
Eunice E. Santos
Department of Electrical Engineering and Computer Science
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015
phone: +1 (610) 758-4517
Fax: +1 (610) 758-6279
e-mail`santos@eecs.lehigh.edu`

EUNICE E. SANTOS is an Assistant Professor in the Department of Electrical Engineering and Computer Science at Lehigh University. She joined the faculty at Lehigh in August of 1995 after receiving a Ph.D. in Computer Science from the University of California, Berkeley in May of that year. Dr. Santos has also obtained B.S. and M.S. degrees in both Mathematics and Computer Science. Dr. Santos's research interests include: parallel and distributed processing, algorithm design and analysis, parallel complexity, scientific and numerical computing, optimization, computational science, and evolutionary computing. She has over 30 technical publications in parallel processing alone. In 1996, Dr. Santos was awarded an NSF CAREER Grant in order to pursue her research interests.