Solving Triangular Linear Systems in Parallel using Substitution

Eunice E. Santos
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720

Abstract

Working within the LogP model, we present parallel triangular solvers which use forward/backward substitution and show that they are optimal. We begin by deriving several lower bounds on execution time for solving triangular linear systems. Specifically, we derive lower bounds in which it is assumed that the number of data items per processor is bounded, a general lower bound, and lower bounds for specific data layouts commonly used for this problem. Furthermore, algorithms are provided which have running times within a constant factor of the lower bounds described. One interesting result is that the popular 2-dimensional block matrix layout necessarily results in significantly longer running times than simpler one-dimensional schemes. Finally, we present a generalization of the lower bounds to banded triangular linear systems.

1 Introduction

This paper deals with designing optimal parallel triangular solvers on distributed-memory machines. Virtually all existing papers deal with designing triangular solvers on particular topologies such as the ring or hypercube [7, 8, 14, 16]. Therefore little has been done on developing lower bounds on the running time for solving triangular linear systems for specific topologies much less developing lower bounds for distributed-memory machines in general. With such lower bounds, we can determine what types of data distributions are needed to achieve efficient running times.

In order to derive upper and lower bounds on the running time for distributed-memory machines in general, we consider a recently proposed model for parallel computation, called the LogP model [3]. LogP has the important feature that the interconnection network of the machine is modeled by its performance as viewed by the user, rather than its detailed interconnection structure. By using the parameters in LogP, many important characteristics of parallel machines can be represented. Algorithms designed on this model are portable from one distributed-memory machine to another and the running times of these algorithms will vary from machine to machine according to the parameter values associated with these machines.

We focus on deriving lower bounds for the time required to solve triangular linear systems and provide algorithms, i.e. triangular solvers, which achieve these bounds all within the LogP model. The bounds we prove are applicable to algorithms which utilize forward/backward substitution. Using these bounds, we are able to determine which types of data layouts should be assumed in order to achieve an optimal running time.

Among other things, we shall show that the communication parameters of a network have a significant effect on the complexity of this problem. We shall also show that optimal algorithms can be obtained using common data layouts and straightforward communication patterns. Of particular interest, we show that block data layout and block cyclic layouts can incur much higher running times than those of many other common data layouts, such as row/column wrapped. Also, we shall see that restricting the proportion of data items assigned to a processor does not result in a significantly higher complexity than assuming that all processors have access to all data items.

The paper is divided as follows. In Section 2 we describe the LogP model. In Sections 3 and 4 we present tight asymptotic bounds for solving triangular systems using forward/backward substitution. Specifically, we present lower bounds on execution time independent of the data layout, lower bounds for data layouts in which the number of data items per processor is bounded, and lower bounds for specific data layouts commonly used in designing parallel algorithms for this problem, including block cyclic layouts of High Performance Fortran [10] and SCALAPACK [6] among others. Furthermore, triangular solvers are provided which have running times within a constant factor of the lower bounds described. Finally, we present a generalization of the lower bounds to banded triangular linear systems. Section 5 gives the conclusion and summary of results.

2 The LogP Model

LogP is a model of a distributed-memory multiprocessor in which processors communicate by point-to-point messages [3]. The model specifies the performance characteristics of the interconnection network, but does not describe the structure of the network. We have tailored the description of the model using terminology specific to the problem of solving triangular linear systems. The main parameters of the model are:

P: the number of processor/memory modules.

L: an upper bound on the latency, or delay, incurred in communicating a message containing a numeri-
cal value from its source module to its target module.

\( o \): the overhead, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform arithmetic operations.

\( g \): the gap, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a processor.¹

Arithmetic operations execute in unit time (a processor cycle). The parameters \( L \), \( o \) and \( g \) are measured as multiples of the processor cycle. Furthermore, it is assumed that the network has a finite capacity, such that at most \( \lfloor L/g \rfloor \) messages can be in transit from any processor or to any processor at any time. If a processor attempts to transmit a message that would exceed this limit, it stalls until the message can be sent without exceeding the capacity limit. All algorithms discussed in this paper satisfy the capacity constraint of the LogP model, and we do not mention it henceforth.

### 2.1 Matching the Model to Real Machines

Transmission of an \( M \)-bit long message in an unloaded or lightly loaded network has four parts in a real machine. First, there is the send overhead; i.e., the time that the processor is busy interfacing to the network before the first bit of data is placed onto the network. Second, there is the time to get the last bit of an \( M \)-bit message into the network having a channel width of \( w \). Third, there is the time for the last bit to cross the network to the destination node which is \( Hr \), where \( H \) is the distance of the route and \( r \) is the delay through each intermediate node. Finally, there is the receive overhead, i.e., the time from the arrival of the last bit until the receiving processor can do something useful with the message. Therefore, the total message communication time for an \( M \) bit message across \( H \) hops is given by the following:

\[
T(M,H) = L_{\text{md}} + \left\lfloor \frac{M}{w} \right\rfloor + Hr + T_{\text{recv}}
\]

It appears reasonable to choose \( o = \frac{L_{\text{md}} + T_{\text{recv}}}{2} \), \( L = Hr + \left\lfloor \frac{M}{w} \right\rfloor \), where \( H \) is the maximum distance of a route and \( M \) is the fixed message size being used, and \( g \) to be \( M \) divided by the per processor bisection bandwidth. Table 1 contains network parameters for various parallel machines.

### 3 Forward/Backward Substitution for Solving Triangular Linear Systems

**The Problem:** Given \( Tx = b \) solve for \( x \), where \( T = (a_{i,j}) \) is a (lower) triangular \( n \times n \) matrix, \( b = (b_j) \) is a vector of size \( n \), and \( x = (x_j) \) is a vector of size \( n \).

We solve triangular linear systems using the forward/backward substitution method. Our decision to focus on this method is based on the fact that while there have been many parallel methods for solving triangular linear systems [1, 9, 12, 17], many of these methods have been shown to be numerically unstable and/or require a number of arithmetic operations which is not optimal [5, 11]. On the other hand, substitution algorithms have been shown to have perfect numerical stability and clearly use the minimum number of arithmetic operations [11]. In addition, substitution is a standard method utilized by algorithm designers.

An algorithm, i.e. triangular solver, which utilizes substitution is one which solves a (lower) triangular system such that each \( x_i \) is evaluated by first evaluating \( x_{i-1}, x_{i-2} \ldots x_1 \) and then substituting these values into the equation for \( x_i \). Clearly any substitution algorithm must have at least \( n^2 \) operations and therefore requires a minimum of \( \frac{n^2}{2} \) time steps.

We now define the class of substitution algorithms formally.

**Definition 3.1** For all \( i \leq n \), a psum (partial sum) of \( x_i \) has the following form:

\[
\tilde{x}_i = \begin{cases} \frac{1}{a_{i,i}}(b_i - \sum_{j=1}^{i-1} a_{i,j} \tilde{x}_j) & \text{if } a_{i,i} \neq 0, \\ \frac{1}{a_{i,i}} \sum_{j=1}^{i-1} a_{i,j} \tilde{x}_j & \text{if } a_{i,i} = 0 \end{cases}
\]

where \( \tilde{x}_i = x_i \) or \( 1 \), \( \tilde{b}_j = b_j \) or 0, and \( \tilde{x}_j = x_j \) or 0.

An algorithm is viewed as a set of arithmetic operations where each processor is assigned a sequential list of these operations. Since we focus on the forward/backward substitution method, we clearly deal only with algorithms whose arithmetic operations create linear combinations (psums) of \( x_j \)'s either by (1) multiplying/dividing data items or (2) using an arithmetic operation on two psums to form a new psum. Below is the formal definition.

**Definition 3.2** We say an algorithm \( A \) is a substitution algorithm if each arithmetic operations is one of the following: (1) multiplying some \( a_{i,j} \) and \( x_j \) where \( j < i \), (2) adding/subtracting two psums of some \( x_i \), resulting in another psum of \( x_i \), or (3) dividing a psum of some \( x_i \) by \( a_{i,i} \) to form another psum of \( x_i \). We denote the class of substitution algorithms by \( S \).

Three components are needed in order to determine running time: the algorithm, the data layout, and the communication pattern. A data layout is the initial assignment of data to the processors. A communication pattern is a list of message transmissions and receptions between processors. It follows from the above definition that processors can only receive or transmit the following values: some \( b_j \) or \( a_{i,j} \) or a psum of some \( x_j \). Moreover, since we are working in the LogP model, we assume that

1. if processor \( p \) transmit a message to \( \tilde{p} \) at time \( t \) then
   (a) it takes \( o \) steps for \( p \) to place the message into the network,
Table 1: Network timing parameters for a one-way message without contention on several current commercial and research multiprocessors. The final row refers to the active message layer, which uses the commercial hardware, but reduces the interface overhead.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Network</th>
<th>Cycle</th>
<th>$\omega$</th>
<th>$T_{\text{snd}} + T_{\text{cu}}$</th>
<th>$r$</th>
<th>avg. $H$</th>
<th>$T(M=160)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ns</td>
<td>bits</td>
<td>cycles</td>
<td>cycles</td>
<td>(1024 Proc.)</td>
<td>(1024 Proc.)</td>
</tr>
<tr>
<td>J-Machine[4]</td>
<td>3d mesh</td>
<td>31</td>
<td>8</td>
<td>16</td>
<td>2</td>
<td>12.1</td>
<td>60</td>
</tr>
<tr>
<td>Monsoon[15]</td>
<td>Butterfly</td>
<td>20</td>
<td>16</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>CM-5 (AM)</td>
<td>Fattree</td>
<td>25</td>
<td>4</td>
<td>132</td>
<td>8</td>
<td>9.3</td>
<td>246</td>
</tr>
</tbody>
</table>

(b) the message is in transit in the network for $L$ steps, and
(c) it takes $o$ steps for $p$ to retrieve the message.

2. If processor $p$ receives (transmits) a message at time $t$, $p$ cannot receive (transmit) another message until time $t+g$ and cannot perform any arithmetic operation until time $t+o$.

All the algorithms and communication patterns we present satisfy all the constraints of the LogP model. Moreover, our lower bounds hold even under the stronger assumption that any value computed by a processor $p$ is available to that processor immediately and to all other processors $L$ steps later.

**Definition 3.3** The class of all communication patterns is denoted $C$. The class of all data layouts is denoted $D$. A data layout $D$ is said to be a single-item layout if each matrix entry $a_{i,j}$ is initially assigned to a unique processor.

**Definition 3.4** For $i = 1, 2, \ldots, n$, define $T_{A,D,C}(x_i)$ to be the time at which $x_i$ is computed using algorithm $A$ and communication pattern $C$ and assuming data layout $D$. (If $x_i$ is computed more than once, $T_{A,D,C}(x_i)$ is the time at which $x_i$ was first computed.)

Since we use substitution algorithms, and $x_{i+1}$ depends on $x_i$, it follows that for all $A \in S$, $D \in D$, $C \in C$, and any $i < n$, $T_{A,D,C}(x_{i+1}) > T_{A,D,C}(x_i)$.

**Definition 3.5** Let $A \in S$ and $D \in D$. For all $i < n$, $T_{A,D}(x_i) = \min_{C \in C} T_{A,D,C}(x_i)$, $T_A(x_i) = \min_{D \in D} T_{A,D}(x_i)$ and $T_S(x_i) = \min_{A \in A} T_A(x_i)$ (i.e. $T_{A,D}(x_i)$ is the minimum time needed to compute $x_i$ using algorithm $A$ and data layout $D$ regardless of communication pattern, $T_A(x_i)$ is the minimum time needed to compute $x_i$ using algorithm $A$ regardless of data layout and communication pattern, and $T_S(x_i)$ is the minimum time needed to compute $x_i$ by any algorithm in the class $S$ regardless of data layout and communication pattern).

In the following sections we shall prove lower bounds on $T_{A,D}(x_i)$ for algorithms $A \in S$ and certain types of data layouts $D$. The lower bounds hold regardless of the choice of communication pattern. For simplicity, we state our results in the special case $L = g$ of the LogP model. In Section 3.1 we prove lower bounds on $S$ assuming $D$ is a single-item data layout in which the number of data items a processor is initially assigned is bounded. In Section 3.2 we assume $D$ is the data layout in which every processor has a copy of every data item (i.e. all $a_{i,j}$ and $b_j$). We present a lower bound on $T_{A,D}(x_i)$. Since $D$ is the most favorable, if impractical, data layout, there is no algorithm $A \in S$ and data layout $D \in D$ which can complete earlier than this bound. Using the bounds obtained, we are able to determine, based on problem size, when a triangular system should be solved using a serial algorithm and when multiple processors would be beneficial. In Section 3.3 we prove some lower bounds for $A \in S$ where $D$ is a standard data layout such as the row/column wrapped data layout. All proofs for this paper are provided in [18]. Although the proofs of the lower bounds in Sections 3.1-3.3 are based on the assumption that $a = 0$, they are clearly applicable to arbitrary $a$. In Section 3.4 we present an extension of the class of algorithms where the lower bounds are still applicable. In Section 3.5, we discuss algorithms for various data layouts. These algorithms all run within a constant factor of the lower bounds shown in this section.

### 3.1 Lower bounds for $A \in S$ where $D$ is a $\frac{P}{n}$ data layout

Many algorithms designed for solving triangular linear systems assume that the data layout is single-item and that each processor is assigned roughly $\frac{P}{n}$ of the data items of $T$ where $P$ is the number of processors available [7, 8, 14, 16]. In this subsection we consider single-item data layouts where each processor can be assigned at most a fraction $\frac{P}{n}$ of the data items of $T$ where $1 \leq c \leq \frac{P}{n}$.

**Definition 3.6** Consider $c$ where $1 \leq c \leq \frac{P}{n}$. A data layout $D$ on $P$ processors is said to be a $\frac{P}{n}$-data layout if $D$ is single-item and no processor is assigned more than a fraction $\frac{P}{n}$ of the $a_{i,j}$'s of $T$. Denote the class of $\frac{P}{n}$ data layouts by $D(\frac{P}{n})$. 

555
Theorem 3.1 If \( D \in D(\frac{P}{2}) \), then for any \( A \in S \),
\[
T_{A,D}(x_n) \geq \max\left(\left\lfloor 1 - \frac{\sqrt{c}}{\sqrt{2}} \right\rfloor n + 1 \right) g, \frac{n^2}{P}.
\]

The complexity of any algorithm \( A \) in our class using a \( \frac{P}{2} \)-data layout is \( \Omega(n g + \frac{n^2}{P}) \); we see that the "communication part" of the bound, grows only linear in problem size \( n \) and gap \( g \) and is independent of \( P(\geq 1) \), whereas the "computation part" grows quadratically in \( n \) and is dependent on \( P \). In addition, although the number of data items assigned to different processors may vary from no data items to \( \frac{1}{2} \) of the total number of data items, the above results and the algorithms provided in Section 3.5 show that the skewness of the distribution of data has no significant effect on the complexity.

Since \( c \leq \frac{n}{P} \), every \( \frac{P}{2} \)-data layout is a \( \frac{1}{2} \)-data layout. This leads to the following corollary:

Corollary 3.1 If \( D \in D(\frac{P}{2}) \), then for any \( A \in S \),
\[
T_{A,D}(x_n) \geq \max\left(\left\lfloor \frac{\sqrt{2} - 1}{\sqrt{2}} \right\rfloor n + 1 \right) g, \frac{n^2}{P}.
\]

3.2 A general lower bound for all \( A \in S \)

In the previous subsection we proved lower bounds for data layouts in which the number of data items assigned per processor is bounded. In this subsection we assume the data layout is the one in which each processor has a copy of every \( a_{i,j} \) and \( b_j \) where \( i, j \leq n \). We denote this layout by \( D \). Since \( D \) is the most favorable data layout, \( T_S(x_n) = \min_{A \in S} T_{A,D}(x_n) \). This layout is interesting even though it is impractical for large matrices because it yields a general lower bound which is not much smaller than the bound in the last section.

Theorem 3.2
\[
T_S(x_n) \geq \left\lfloor \frac{n^2}{P} \right\rfloor \text{ if } n \leq \left\lfloor \sqrt{g} \right\rfloor \quad \text{max}\left(\left\lfloor \frac{n^2+4g-n}{g} \right\rfloor, \frac{n^2}{P} \right) \text{ if } n \geq g
\]

Comparing the bounds from Sections 3.1-3.2 and considering the algorithms in Section 3.5, we see that for \( n \geq g \), the restriction to \( \frac{P}{2} \)-data layouts increases complexity by only a constant factor. Also, we observe that when \( n \leq g \), a parallel algorithm will not be much more beneficial than using a serial algorithm.

3.3 Lower Bounds for \( S \) on standard data layouts

We present lower bounds on the running time for algorithms using specific data layouts which are commonly used by algorithm designers. The data layouts we consider are the following: row/column wrapped, row/column contiguous, block decomposition, and block-cyclic. All of these are standard layouts in High Performance Fortran [10] and SCALAPACK [6] among others.

![Figure 1: Data allocation for Standard Data Layouts](image)

Formal definitions of these data layouts are given below. Also, Figure 1 shows the data allocation to processors for each data layout (excluding block cyclic) with \( n = 12, P = 3 \). For block cyclic layout we assume that \( n = 16, P = 4 \), and \( K = 4 \), each entry in the matrix denotes which processor is assigned which data item.

Definition 3.7 The row(column) wrapped data layout on \( P(\leq n) \) processors \( p_1, p_2, \ldots, p_P \) is defined as follows: for all \( i \leq n, a_{i,1}, \ldots, a_{i,i} \) is assigned to processor \( p_j \) where \( j + 1 \equiv i \mod P \). We denote this layout by \( D_{RW}(D_{CW}) \).

Definition 3.8 A single-item data layout on \( P = \frac{P(\leq n)}{2} \) (i.e. \( P = \frac{\sqrt{1+4P^2}}{2} \leq n \)) processors is said to be a block data layout if each processor is given a
Definition 3.9 A single-item data layout on $P(\leq n^2)$ processors is said to be a (square) block-cyclic data layout if there exist a $1 \leq K \leq \frac{n^2}{\sqrt{P}}$ : $T$ is divided into contiguous square blocks of size $\frac{n^2}{K \sqrt{P}}$. Furthermore, each such block is divided into smaller contiguous square sub-blocks of size $\frac{n^2}{K \sqrt{P}}$. Each processor is then assigned a sub-block of each block of size $\frac{n^2}{K \sqrt{P}}$ such that each sub-block is in the same position for each block. We denote this data layout by $D_{K,BC}$.

Definition 3.10 A single-item data layout on $P(\leq n^2)$ processors $p_1, \cdots, p_P$ is row(column) contiguous if for all $i \leq P$, $p_i$ is assigned the matrix items in rows (columns) $(i-1)\frac{n^2}{P} + 1$ to $i\frac{n^2}{P}$. We denote this layout by $D_{K,R}(D_{K,C})$.

We show that block decomposition and block-cyclic data layouts have higher lower bounds than row/column wrapped or row/column contiguous.

Theorem 3.3 The lower bounds presented in Table 2 hold for any substitution algorithm.

Analyzing the lower bounds in Table 2, we see that row/column wrapped and row/column contiguous have lower bounds of $\Omega(n^2)$. We have shown that the lower bound for block data layout is $\Omega(n^2)$. Clearly

$$\max(ng, \frac{n^2}{P}) < \max(ng, \frac{n^2}{P}, \min(\frac{n^2}{Kg}, \frac{n^2}{\sqrt{P}}))$$

only when $P < \min(n, \frac{n^2}{P})$. The lower bound for block cyclic layout is

$$\Omega(\max(n^2, \min(\frac{n^2}{P}, \frac{n^2}{\sqrt{p}}))).$$

Again, clearly

$$\max(ng, \frac{n^2}{P}) < \max(ng, \frac{n^2}{P}, \min(\frac{n^2}{Kg}, \frac{n^2}{\sqrt{P}}))$$

when $K < \min(\sqrt{P}, \frac{n^2}{P}, \frac{n}{\sqrt{P}})$.

The lower bounds are presented for the parameters associated with the CM-5 in Figure 2. The unbroken line represents the bounds for row/column wrapped, row/column contiguous and block cyclic for $K \geq max(\sqrt{P}, \frac{n^2}{P}, \frac{n}{\sqrt{P}})$. The broken line represents the bound for block-cyclic with $K = 1$ and the dotted line represents the bound for block data layout.

3.4 Extended Class of Algorithms $S'$

In this subsection we define an extended class of algorithms $S' \supseteq S$ which we call no-cost inference algorithms. The class $S'$ contains all algorithms in which a processor is allowed to infer the value of a variable $x_i$ from the psums it has computed or received and from data items in its local memory. Such an inference is possible if and only if $x_i$ can be expressed as a rational combination of such psums and data items. When proving lower bounds on complexity we do not charge any cost for such an inference (see Example 3.1). This larger and apparently more powerful class of algorithms turns out to satisfy the same lower bounds as the class $S$.

Example 3.1 Suppose $s_1$ and $s_2$ are psums received or computed by processor $p$ where

$s_1 = b_{i_1} - a_{i_1,j_1} x_{j_1} - a_{i_1,j_2} x_{j_2}$, and

$s_2 = a_{i_2,j_1} x_{j_1} + a_{i_2,j_2} x_{j_2}$.

If $b_{i_1}, a_{i_1,j_1}, a_{i_1,j_2}, a_{i_2,j_1}, a_{i_2,j_2}$ are in the local memory of $p$ then we assume that $p$ can infer (at no cost) the values of $x_{j_1}$ and $x_{j_2}$.

3.5 Algorithms and Communication Patterns

In this section we provide running times of algorithms and communication patterns where when used with the appropriate data layout are within a constant factor of the appropriate lower bounds. The algorithms and communication patterns are provided in [18].

Table 3 provides achievable running times for substitution algorithms, communication patterns and the data layouts listed. The precise running times for block and block-cyclic data layouts are given in Tables 4-5.
Table 2: Lower bounds for standard data layouts.

<table>
<thead>
<tr>
<th>Running Time</th>
<th>Data Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P \sum_{j=1}^{g} \max(g, 2j + o + 2) + (n - 1)(2o + g + 1) = O(ng + \frac{n^2}{P}) )</td>
<td>Row Wrapped</td>
</tr>
<tr>
<td>( \frac{P}{n^2} + \frac{3(n - 1)}{P} + \max(\frac{n^2}{P}, \frac{n^2}{K^2}, (n - 1)(2o + g + 1)) )</td>
<td>Column Wrapped</td>
</tr>
<tr>
<td>( (\frac{n}{P})^2 + \frac{P}{n^2} \max(\frac{n^2}{P} + o + 2, g) + (3P - 4o + (P - 1)g + P - 2) )</td>
<td>Contiguous Row</td>
</tr>
<tr>
<td>( \max(\frac{n^2}{P}, \frac{n^2}{K^2}, g) )</td>
<td>Contiguous Column</td>
</tr>
<tr>
<td>( O(ng + \frac{n^2}{P}) )</td>
<td>Contiguous Column</td>
</tr>
<tr>
<td>( O(ng + \frac{n^2}{P}) )</td>
<td>Block Layout</td>
</tr>
<tr>
<td>( O(ng + \frac{n^2}{P}) )</td>
<td>Block Cyclic</td>
</tr>
<tr>
<td>( n^2 ) if ( n &gt; g )</td>
<td>Data Layout ( \tilde{D} )</td>
</tr>
<tr>
<td>( (\frac{n}{P})^2 + \frac{P}{n^2} \max(\frac{n^2}{P} + o + 2, g) + (3P - 4o + (P - 1)g + P - 2) )</td>
<td>Data Layout ( \tilde{D} )</td>
</tr>
</tbody>
</table>

Table 3: Running times for standard data layouts.

<table>
<thead>
<tr>
<th>Running Time</th>
<th>Data Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P \sum_{j=1}^{g} \max(g, 2j + o + 2) + (n - 1)(2o + g + 1) = O(ng + \frac{n^2}{P}) )</td>
<td>Row Wrapped</td>
</tr>
<tr>
<td>( \frac{P}{n^2} + \frac{3(n - 1)}{P} + \max(\frac{n^2}{P}, \frac{n^2}{K^2}, (n - 1)(2o + g + 1)) )</td>
<td>Column Wrapped</td>
</tr>
<tr>
<td>( (\frac{n}{P})^2 + \frac{P}{n^2} \max(\frac{n^2}{P} + o + 2, g) + (3P - 4o + (P - 1)g + P - 2) )</td>
<td>Contiguous Row</td>
</tr>
<tr>
<td>( \max(\frac{n^2}{P}, \frac{n^2}{K^2}, g) )</td>
<td>Contiguous Column</td>
</tr>
<tr>
<td>( O(ng + \frac{n^2}{P}) )</td>
<td>Contiguous Column</td>
</tr>
<tr>
<td>( O(ng + \frac{n^2}{P}) )</td>
<td>Block Layout</td>
</tr>
<tr>
<td>( O(ng + \frac{n^2}{P}) )</td>
<td>Block Cyclic</td>
</tr>
<tr>
<td>( n^2 ) if ( n &gt; g )</td>
<td>Data Layout ( \tilde{D} )</td>
</tr>
<tr>
<td>( (\frac{n}{P})^2 + \frac{P}{n^2} \max(\frac{n^2}{P} + o + 2, g) + (3P - 4o + (P - 1)g + P - 2) )</td>
<td>Data Layout ( \tilde{D} )</td>
</tr>
</tbody>
</table>

Clearly, since \( o \leq g \), all of these running times are within a constant factor of the corresponding lower bounds. Furthermore, since the algorithms are variants of standard algorithms using simple communication patterns, it follows that employing sophisticated techniques will not yield a significant improvement in the running time.

Another point worth mentioning is that block layouts with \( P < \min(n, \frac{n^2}{P}) \) and block-cyclic layouts with \( K < \min(V, \frac{n}{P}, \frac{n}{K}) \) incur much higher running times than the other layouts. In general, these two data layouts are considered for this problem because they are standard layouts for LU decomposition [5]. Therefore it may be possible to mask the extra running times by the running time needed for LU decomposition. However, if one simply needs to solve \( Tx = b \), it will probably be better to employ one of the other data layouts or to choose appropriate values of \( P \) and \( K \) if using either block decomposition or block cyclic.

4 Banded Triangular Linear Systems

In this section, we generalize the lower bounds presented in the previous subsections from lower bounds for solving triangular systems to lower bounds for solving banded triangular systems. We again consider only substitution algorithms.

The Banded Problem: Given \( Tx = b \) solve for \( x \), where \( T = (a_{ij}) \) is a \( k \)-banded (lower) \( n \times n \) triangular matrix, \( b = (b_1) \) is a vector of size \( n \), and \( x = (x_j) \) a vector of size \( n \).

The serial complexity is \( 2nk - k^2 \).

We now generalize some of the previous definitions.

Definition 4.1 Let \( D \) be a single-item data layout on \( P \) processors. Let \( 1 \leq c \leq \frac{P}{k} \). If no processor is as-
signed more than a $\frac{c}{x}$ fraction of the items of $T$ then $D$ is a $\frac{1}{x}$-data layout for $k$-banded triangular matrix $T$. We refer to the set of $\frac{1}{x}$ - data layouts as $D(\frac{1}{x}, k)$.

Definition 4.2 Let $D \in D(\frac{1}{x}, k)$. Define $i_{n,k,x}$ to be the smallest integer $i$ such that the number of non-zero items in row 1 to row $i$ of $T$ is greater than $\frac{2nk^2+k\cdot c}{x}$.

It follows immediately that $i_{n,k,x} \leq \sqrt[2]{x} \cdot n$.

Definition 4.3 Define $D_k$ to be the data layout in which every processor is assigned every matrix item in $T$ and every item in $b$.

Theorem 4.1 For any $A$ in the class $S$,

$$T_S(x_n) \geq \left\{ \begin{array}{ll} \frac{2}{k^2} k^2 & \text{if } k \leq \sqrt[2]{x} \\ \frac{2}{k^2} \left[k^2+\sqrt[2]{x} - 3\right] & \text{if } \sqrt[2]{x} \leq k \leq g \\ \max\left(\left\{ \frac{2}{k^2} \left[k^2+\sqrt[2]{x} - 3\right], \frac{2}{k^2} \right\}, \frac{2}{k^2} \right) & \text{if } k \geq g \end{array} \right.$$

Theorem 4.2 Let $A \in S$ and $D \in D(\frac{1}{x}, k)$. If $k \geq g$ then

$$T_{A,D}(x_n) \geq \max((n - \sqrt[2]{x} \cdot n + 1)g, \frac{nk}{P}).$$

Since $c \leq \frac{P}{x}$, every $\frac{1}{x}$-data layout is a $\frac{1}{x}$-data layout. This leads to the following corollary:

**Corollary 4.1** If $D \in D(\frac{1}{x}, k)$, then for any $A \in S$,

$$T_{A,D}(x_n) \geq \max((\frac{\sqrt[2]{x} - 1}{2} n + 1)g, \frac{nk}{P}).$$

We see that for $n > g$, the lower bound is $\Omega(n^2 + \frac{nk}{P})$.

5 Conclusion

In this paper we considered the problem of solving triangular linear systems on parallel distributed-memory machines using substitution. Working within the LogP model [3], we were able to derive tight asymptotic bounds on the execution time for this problem and provided algorithms which achieve these bounds.

Specifically, we proved that for sufficiently large matrices ($n \geq g$), the running time of any substitution algorithm is $\Omega(n^2 + \frac{nk}{P})$ whereas the serial complexity is $n^2 + O(n)$. When we then restrict attention to data layouts in which the number of data items assigned to a processor is bounded, the lower bound is still $\Omega(n^2 + \frac{nk}{P})$. Since these bounds are achievable, this shows that simply restricting the proportion of data items assigned to a processor does not result in a significantly higher complexity than assuming all processors have all the data items.

We also derived bounds for several specific data layouts which are commonly used by algorithm designers.
for this problem. In particular, we showed that blockcyclic data layouts with \( K < \min(\sqrt{P}, \frac{1}{\sqrt{P}}, \frac{n^2}{g}, \frac{g}{n^2}) \) and block data layout with \( P < \min(n, \frac{n^2}{g}) \) have much higher complexities than all the other data layouts we considered.

We showed that there exist substitution algorithms for these data layouts whose running times are within constant factors of the corresponding lower bounds. Since all of these were variants of standard algorithms using simple communication patterns, this shows that utilizing sophisticated techniques, and designing communication patterns which greatly minimize communication between processors, yield no significant benefits toward the running time of a substitution algorithm.

Lastly, we generalize the problem to \( k \)-banded triangular linear systems. We showed that for \( k \geq g \), the running time is \( \Omega(nk + \frac{n^2}{g}) \). Therefore, as before, we see that designing sparse communication patterns gives no significant benefits.

All lower bounds obtained in this paper hold for an extended model with multi-broadcast capability, i.e. the lower bounds hold even under the assumption that any value computed by a processor \( p \) is available to that processor immediately and to all other processors \( L \) steps later.

Acknowledgements

I would like to especially thank Richard M. Karp and James Demmel for all the discussions and helpful comments they gave me on this paper.

Research was supported by a DoD-NDSEG Graduate Fellowship and NSF Grant CCR-90-17380.

References


