Parallel Solution of Block Tridiagonal Linear Systems

Roberto Bevilacqua,
Dipartimento di Informatica
Corso Italia 40
56100 Pisa, Italy

and

Bruno Codenotti and Francesco Romani
IEI-CNR
Via S. Maria 46
56100 Pisa, Italy

Submitted by Graciano de Oliveira

ABSTRACT

The explicit structure of the inverse of block tridiagonal matrices is presented in terms of blocks defined by linear recurrence relations. Parallel algorithms are shown which solve block second order linear recurrences without using commutativity. Moreover we investigate the parallel solution of the associated block tridiagonal linear system. Using this theoretical background, the implementation of the algorithms is analyzed both on a small number of processors and on a hypercube. The resulting complexity is given in terms of parallel steps, each consisting of block operations, and the cost due to interprocessor communications is taken into account, too.

1. INTRODUCTION

Both the problems of characterizing the structure of the inverse of a block tridiagonal matrix and of devising efficient parallel algorithms for the solution of the associated linear system have been extensively studied [1, 6, 10, 16]. Recently special attention has been given to the implementation of solving algorithms on vector and parallel computers [2, 7, 8, 14, 15]. The aim of this paper is to present the explicit structure of the inverse and to derive efficient parallel algorithms.
Let us consider a square nonsingular real matrix $A$ of order $nm \times nm$ partitioned into an $n \times n$ matrix of square blocks. These blocks, denoted by $A_{ij}$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$, are $m \times m$ matrices. When $A_{ij} = 0$ for $j - i > 1$, the matrix $A$ is said to be block upper Hessenberg; when $A_{ij} = 0$ for $i - j > 1$, the matrix $A$ is said to be block lower Hessenberg; when $A_{ij} = 0$ for $|i - j| > 1$, the matrix $A$ is said to be block tridiagonal. In Section 2, the structure of the inverse of block Hessenberg and block tridiagonal matrices is presented, assuming that subdiagonal (or superdiagonal) blocks are nonsingular. Matrices of this kind are said to be proper. The inverse $B$ of a proper nonsingular block tridiagonal matrix can be written as

$$B_{ij} = \begin{cases} -F_i S^{-1} G_j & \text{if } i \leq j, \\ -E_i R^{-1} D_j & \text{if } i > j, \end{cases}$$

where the blocks $D_i, E_i, F_i, G_i, R, S$ are defined by second order linear recurrence relations.

In Section 3, we discuss parallel algorithms which solve block second order linear recurrences without using commutativity. Regardless to communication costs and assuming the processors to be capable of performing $m \times m$ block operations in time $B$, the resulting algorithms have time complexity $O(B \log n)$.

In Section 4, algorithms for computing the inverse of a block tridiagonal matrix and the solution of the associated linear system are presented.

In Section 5, the implementation of the algorithms on a small number of processors is studied, taking into account communication costs and the different costs of the various block operations.

Finally, in Section 6, the implementation on a set of $n$ processors connected by a hypercube network [9, 12, 13] is studied; moreover the complexity of the implementation on a perfect shuffle [8, 12, 13] is analyzed, in order to compare our results with those shown in [8].

2. THE INVERSE OF BLOCK HESSENBERG AND TRIDIAGONAL MATRICES

The structure of the upper (lower) triangular part of the inverse of block lower (upper) Hessenberg matrices is presented, assuming that subdiagonal (superdiagonal) blocks are nonsingular. Using this result, the explicit structure of the inverse of block tridiagonal matrices is derived. Commutativity of blocks is not required.
In the following we give the definition of a proper Hessenberg and tridiagonal matrix. It is worth noting that such definitions are equivalent to that of a proper banded matrix used in [11].

**Definition 2.1.** An $n \times n$ block lower (upper) Hessenberg matrix is said to be proper if all the superdiagonal (subdiagonal) blocks are nonsingular.

**Definition 2.2.** An $n \times n$ block tridiagonal matrix is said to be proper if all the superdiagonal and subdiagonal blocks are nonsingular.

**Definition 2.3.** Let $H$ be the $n \times n$ block lower Hessenberg matrix

$$
H = \begin{bmatrix}
A_{11} & A_{12} & 0 & 0 & \cdots & 0 \\
A_{21} & A_{22} & A_{23} & 0 & \cdots & 0 \\
& \ddots & \ddots & \ddots & \vdots & \vdots \\
A_{n-1,1} & A_{n-1,2} & A_{n-1,3} & \cdots & A_{n-1,n} \\
A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn}
\end{bmatrix},
$$

with $A_{ij}$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$, $m \times m$ matrices. Let $I_p$ denote the identity matrix of size $p$. We denote by $H'$ the $(n + 1) \times (n + 1)$ block lower triangular matrix defined as

$$
H' = \begin{bmatrix}
I_m & 0 & 0 & 0 & \cdots & 0 & 0 \\
A_{11} & A_{12} & 0 & 0 & \cdots & 0 & 0 \\
A_{21} & A_{22} & A_{23} & 0 & \cdots & 0 & 0 \\
& \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\
A_{n-1,1} & A_{n-1,2} & A_{n-1,3} & \cdots & A_{n-1,n} & 0 \\
A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nn} & I_m
\end{bmatrix}.
$$
**Definition 2.4.** Let \( H \) be an \( n \times n \) block upper Hessenberg matrix. We denote by \( H'' \) the \((n + 1) \times (n + 1)\) block upper triangular matrix

\[
H'' = \begin{bmatrix}
I_m & H \\
0 & 0 \\
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & I_m
\end{bmatrix}.
\]

**Definition 2.5.** Let \( T \) be an \( n \times n \) block tridiagonal matrix whose blocks, denoted by \( A_{ij}, \ i = 1, 2, \ldots, n, \ j = 1, 2, \ldots, n, \) are \( m \times m \) matrices. We denote by \( T' [T''] \) the \((n + 1) \times (n + 1)\) block lower [upper] triangular matrices

\[
T' = \begin{bmatrix}
I_m & 0 & \ldots & 0 & 0 \\
0 & T & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \ldots & I_m
\end{bmatrix},
\]

\[
T'' = \begin{bmatrix}
I_m & T \\
0 & 0 \\
\vdots & \vdots \\
0 & 0 & \ldots & I_m
\end{bmatrix}.
\]

Note that \( H', H'', T', T'' \) are nonsingular if and only if the related matrices \( H \) and \( T \) are proper.

**Proposition 2.1.** Let \( H \) be a nonsingular, proper, block lower Hessenberg matrix, and let \( H' \) be its associated block lower triangular matrix. Partition \( H'^{-1} \) as

\[
\begin{bmatrix}
F & Z' \\
S & G
\end{bmatrix},
\]

where \( F, S, Z', \) and \( G \) are block matrices of size \( n \times 1, 1 \times 1, n \times n, 1 \times n, \) respectively. Then \( H^{-1} = -FS^{-1}G + Z' \).
Proof. We have

\[
HF = \begin{bmatrix}
0 \\
\vdots \\
0 \\
S
\end{bmatrix}, \quad HZ' = I_{m_n} - \begin{bmatrix}
0 \\
\vdots \\
0 \\
G
\end{bmatrix}.
\]

Let \( Q \) be an \( 1 \times n \) block matrix; then

\[
H(FQ + Z') = I_{n_m} - \begin{bmatrix}
0 \\
\vdots \\
0 \\
SQ + G
\end{bmatrix}.
\]

From a theorem of Jacobi (see, [5, p. 14]), it follows that

\[
|\det H| = |\det H'||\det S|,
\]

whence \( \det S \neq 0 \), and the thesis follows by choosing \( Q = -S^{-1}G \).

**Proposition 2.2.** Let \( H \) be a nonsingular, proper, block upper Hessenberg matrix, and let \( H'' \) be its associated block upper triangular matrix. Partition \( H^{-1} \) as

\[
\begin{bmatrix}
D & R \\
Z'' & E
\end{bmatrix},
\]

where \( E, R, Z'', \) and \( D \) are block matrices of size \( n \times 1, 1 \times 1, n \times n, 1 \times n \), respectively. Then \( H^{-1} = -ER^{-1}D + Z'' \).

Proof. The proof is analogous to that of Proposition 2.1.

In the following it will be useful to represent the matrices \( D, F, E, G \) as block vectors, i.e.

\[
D = [D_1, D_2, \ldots, D_n], \quad G = [G_1, G_2, \ldots, G_n],
\]

\[
E = \begin{bmatrix}
E_1 \\
E_2 \\
\vdots \\
E_n
\end{bmatrix}, \quad F = \begin{bmatrix}
F_1 \\
F_2 \\
\vdots \\
F_n
\end{bmatrix}.
\]
With this notation it is easy to characterize $D, E, F, G, R, S$ by means of recurrence relations, and to derive the inverse of a proper block tridiagonal matrix, as shown in the following propositions.

**Proposition 2.3.** The matrices $D_i, E_i, F_i, G_i, R, S$ can be obtained by means of the following relations:

\[
D_1 = I, \quad D_2 = -A_{11}^{-1} A_{21}^{-1},
\]
\[
D_i = -(D_{i-2} A_{i-2,i-1} + D_{i-1} A_{i-1,i-1}^{-1}) A_{i,i-1}^{-1}, \quad i = 3, 4, \ldots, n,
\]
\[
R = -(D_{n-1} A_{n-1,n} + D_n A_{n,n}).
\]
\[
E_n = I, \quad E_{n-1} = -A_{n,n-1}^{-1} A_{n,n},
\]
\[
E_i = -A_{i+1,i}^{-1} (A_{i+1,i+1} E_{i+1} + A_{i+1,i+2} E_{i+2}), \quad i = n-1, n-2, \ldots, 1.
\]
\[
F_1 = I, \quad F_2 = -A_{12}^{-1} A_{11},
\]
\[
F_i = -A_{i-1,i}^{-1} (A_{i-1,i-2} F_{i-2} + A_{i-1,i-1} F_{i-1}), \quad i = 3, 4, \ldots, n,
\]
\[
S = -(A_{n,n-1} F_{n-1} + A_{n,n} F_n).
\]
\[
G_n = I, \quad G_{n-1} = -A_{n,n}^{-1} A_{n-1,n},
\]
\[
G_i = -(G_{i+1} A_{i+1,i+1} + G_{i+2} A_{i+2,i+1}) A_{i,i+1}^{-1}, \quad i = n-1, n-2, \ldots, 1.
\]

**Proof.** The proof readily follows from Propositions 2.1 and 2.2.

**Proposition 2.4.** Let $T$ be a nonsingular proper block tridiagonal matrix, and let $T', T''$ be its associated block triangular matrices. Partition $T'^{-1}$ and $T''^{-1}$ as in Propositions 2.1 and 2.2. Let $T^{-1} = (B_{ij})$ with $B_{ij} m \times m$ matrices. Then

\[
B_{ij} = \begin{cases} 
-F_j S^{-1} G_j & \text{if } i \leq j, \\
-E_i R^{-1} D_j & \text{if } i \geq j.
\end{cases}
\]

**Proof.** From Propositions 2.1 and 2.2, the equality

\[
T^{-1} = -FS^{-1} G + Z' = -ER^{-1} D + Z''
\]
can be derived. The proof follows from the fact that $Z'$ ($Z''$) has null blocks in its upper (lower) triangular part.

**Corollary 2.1.** Let $T$ be a proper block tridiagonal matrix. Then the matrices $F_i$ and $D_i$ are nonsingular for any $1 \leq i \leq n$ if and only if the $i \times i$ block minor in the upper left corner of $T$ is nonsingular. Analogously, the matrices $E_i$ and $G_i$ are nonsingular if and only if the $i \times i$ block minor in the lower right corner of $T$ is nonsingular.

**Proof.** The proof follows by applying Propositions 2.1 and 2.2 to selected submatrices of $T$. ■

### 3. SOLVING SECOND ORDER LINEAR RECURRENCES

In this section, we present parallel algorithms for the computation of the solution of some well-known recurrence equations [3]. The notation of [14] will be used in the following.

Let $P_{ik}$ and $Q_{ik}$ be $m \times m$ matrices defined by the second order linear recurrences

\[
P_{ik} = U_k P_{i-1,k-1} + V_k P_{i-2,k-2}, \quad i = 2, 3, \ldots, n, \quad k = 2, 3, \ldots, n,
\]

\[
Q_{ik} = Q_{i-1,k-1} U_k + Q_{i-2,k-2} V_k, \quad i = 2, 3, \ldots, n, \quad k = 2, 3, \ldots, n,
\]

under the following assumptions:

\[
P_{ik} = Q_{ik} = U_k, \quad k = 1, 2, \ldots, n,
\]

\[
P_{ik} = Q_{ik} = I_m, \quad i = 0 \text{ or } k = 0,
\]

where $U_i, V_i, i = 1, 2, \ldots, n$, are $m \times m$ matrices. Matrices $P_{ik}$ and $Q_{ik}$ also satisfy other recurrences than those appearing in their definition, as stated by the following proposition.

**Proposition 3.1.** The following second order linear recurrences hold, independently of commutativity:

\[
P_{i+j,k} = P_{ik} P_{j,k-i} + P_{i-1,k} V_{k-i} + P_{j-1,k-i-1}, \quad i \geq 1, \quad j \geq 1, \quad k \geq i + 1,
\]

\[
Q_{i+j,k} = Q_{i,k-j} Q_{j,k} + Q_{i-1,k-j-1} V_{k-j} + Q_{j-1,k}, \quad i \geq 1, \quad j \geq 1, \quad k \geq j + 1.
\]

**Proof.** See [3, 14]. ■
Our goal is to compute the matrices $P_{ii}, Q_{ii}, \ i = 2, 3, \ldots, n$. In the following only the computation of $P_{ii}, \ i = 2, 3, \ldots, n$ will be treated. The computation of the matrices $Q_{ii}$ can be carried out with the same complexity, by means of straightforward modifications of the same algorithms.

Two algorithms are presented, which compute the matrices $P_{ii}, \ i = 2, 3, \ldots, n$, in $O(\Bbb{B} \log n)$ parallel steps, on $O(n)$ processors, each performing $m \times m$ matrix operations in time $\Bbb{B}$. Both algorithms use the recursive doubling technique applied to the recurrence relations of Proposition 3.1: the former generalizes to the block case the algorithms proposed in [14] for the $LU$ factorization of a tridiagonal matrix, and is slightly different from the recursive doubling algorithm in [2]; the latter has to be regarded as a modification of the former, as suggested in [15] to lower the total number of matrix operations which turns out to be $O(n \log n)$ for the first algorithm, and $O(n)$ for the second one.

Algorithm 3.1.

Step 1:

\[
P_{2k} = U_k U_{k-1} + V_{k-1}, \quad k = 2, 3, \ldots, n.
\]

Step 2:

\[
P_{4k} = P_{2k} P_{2,k-2} + U_k V_{k-1} U_{k-3}, \quad k = 4, 5, \ldots, n;
\]

\[
P_{3k} = P_{2k} U_{k-2} + U_k V_{k-1}, \quad k = 3, 4, \ldots, n.
\]

Step $i$, $i = 3, 4, \ldots, \log n - 2$:

\[
s = 2^{i-1};
\]

\[
P_{2s} = P_{2s} P_{s,k-s} + P_{s-1,k-s+1} V_{k-s+1} P_{s-1,k-s-1}, \quad k = 2s, 2s+1, \ldots, n;
\]

\[
P_{2s-1,k-s} = P_{2s} P_{s-1,k-s} + P_{s-1,k-s+1} V_{k-s+1} P_{s-2,k-s-1}, \quad k = 2s-1, 2s, \ldots, n;
\]

\[
P_{2s-2,k} = P_{s-1,k} P_{s-1,k-s+1} V_{k-s+1} P_{s-2,k-s} - P_{s-2,k-s-1}, \quad k = 2s-2, 2s-1, \ldots, n;
\]

\[
P_{2s-j,2s-j} = P_{s-1,k} P_{s-1,k-s+1} V_{k-s+1} P_{s-j-1,j-1}, \quad j = 3, 4, \ldots, s-1.
\]
Step $\log n - 1$:

$$P_{n/2,k} = P_{n/4,k}P_{n/4,k} + P_{n/4-1,k}P_{n/4+1,k}P_{n/4-1,k-n/4-1},$$

$$k = \frac{n}{2}, \frac{n}{2} + 1, \ldots, n;$$

$$P_{n/2-1,k} = P_{n/4,k}P_{n/4-1,k} + P_{n/4-1,k}P_{n/4+1,k}P_{n/4-2,k-n/4-1},$$

$$k = \frac{n}{2} + 1, \frac{n}{2} + 2, \ldots, n;$$

$$P_{n/2-j,n/2-j} = P_{n/4,n/2-j}P_{n/4-j,n/4-j} + P_{n/4-j,n/2-j}V_{n/4-j+1}P_{n/4-j-1,n/4-j-1},$$

$$j = 1, 2, \ldots, \frac{n}{4} - 1.$$

Step $\log n$:

$$P_{n-j,n-j} = P_{n/2,n-j}P_{n/2-j,n/2-j} + P_{n/2-1,n-j}V_{n/2-j+1}P_{n/2-j-1,n/2-j-1},$$

$$j = 0, 1, \ldots, \frac{n}{2} - 1.$$

Let $\mathcal{P}$ be the number of processors involved in the computations of the matrix recurrence relation. To exploit the inherent parallelism of the algorithm, $(3n - 17)\mathcal{P}$ processors suffice; indeed, the largest number of processors is required at step 3, where the recurrence relation has to be used $3n - 17$ times.

The total number of such computations is $3(n + 1)\log n - \frac{17}{2}n + 2$, so that we have to perform roughly $9n \log n$ matrix multiplications and $6n \log n$ matrix additions.

The scheme of computation for $n = 32$ is described in Table 1.

**Algorithm 3.2.**

Step 1:

$$P_{2,j} = U_{2,j}U_{2,j-1} + V_{2,j-1}, \quad j = 1, 2, \ldots, n/2;$$
Step $i$, $i = 2, 3, \ldots, \log n - 1$:

$$s = 2^{i-1};$$

$$P_{2s, 2ks} = P_{s, 2ks} P_{s, 2(k-1)s} + P_{s-1, 2ks} V_{2(k-1)s} + 1 P_{s-1, 2(k-1)s-1},$$

$$P_{2s-1, 2ks-1} = P_{s-1, 2ks-1} P_{s-1, 2(k-1)s} + P_{s-2, 2ks-1} V_{2(k-1)s} + 1 P_{s-2, 2(k-1)s-1},$$

$$k = 1, 2, \ldots, n/2^i;$$

$$P_{2s-2, 2ks-1} = P_{s-2, 2ks-1} P_{s-1, 2(k-1)s} + P_{s-2, 2ks-1} V_{2(k-1)s} + 1 P_{s-2, 2(k-1)s-1},$$

$$k = 2, 3, \ldots, n/2^i.$$

Step $\log n$:

$$P_{nn} = P_{n/2, n} P_{n/2, n/2} + P_{n/2-1, n} V_{n/2+1} P_{n/2-1, n/2-1},$$

$$P_{n-1, n-1} = P_{n/2-1, n} P_{n/2-2, n/2} + P_{n/2-2, n} V_{n/2+1} P_{n/2-1, n/2-1},$$

$$P_{2n, 2n} = P_{n/4, n} P_{n/4, n/2} + P_{n/4-1, n} V_{n/2+1} P_{n/2-1, n/2-1},$$

$$P_{2n-1, 2n-1} = P_{n/4-1, n} P_{n/4-2, n/2} + P_{n/4-2, n} V_{n/2+1} P_{n/2-1, n/2-1}.$$

Step $\log n + i$, $i = 1, 2, \ldots, \log n - 3$:

$$s = 2^{\log n - i - 2};$$

$$P_{(2k+1)s, (2k+1)s} = P_{s, (2k+1)s} P_{2ks, 2ks} + P_{s-1, (2k+1)s} V_{2ks+1} P_{2ks-1, 2ks-1},$$

$$P_{(2k+1)s-1, (2k+1)s-1} = P_{s-1, (2k+1)s-1} P_{2ks, 2ks} + P_{s-2, (2k+1)s-1} V_{2ks+1} P_{2ks-1, 2ks-1},$$

$$k = 1, 2, \ldots, 2^{i+1} - 1.$$

In this case $(n - 2) n^0$ processors suffice to exploit the inherent parallelism of the algorithm. The total number of computations of the recurrence is $\frac{7}{2} n^2 - 4 \log n - 2$, so that roughly $\frac{21}{2} n$ matrix multiplications and $7n$ matrix additions have to be performed.

The scheme of computation for $n = 32$ is described in Table 2.
TABLE 1

<table>
<thead>
<tr>
<th>Step at which $P_k$ is computed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
</tbody>
</table>

$k = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32$

*Case $n = 32$, 5 steps, 225 computations of the recurrent formula.

4. ALGORITHMS

Propositions 2.3 and 2.4 suggest a straightforward algorithm to compute the inverse of a proper block tridiagonal matrix.

Algorithm 4.1 (Computation of the inverse). Compute $D_1, E_1, F_1, G_1, R, S$ by means of the relations shown in Proposition 2.3.

Let us consider now the linear system $Tx = b$. Partition $x$ and $b$ as $x = [x_1, x_2, \ldots, x_n]$ and $b = [b_1, b_2, \ldots, b_n]$, where $x_i$ and $b_i$ are $m$-vectors. The
solution of the linear system can be written as

\[ x_i = -E_iR^{-1}\sum_{j=1}^{i-1}D_jb_j - F_iS^{-1}\sum_{j=1}^{n}G_jb_j, \quad i = 1, 2, \ldots, n. \]

**ALGORITHM 4.2 (Solving the linear system).**

Compute

\[ R^{-1} \quad \text{and} \quad S^{-1}. \]
Compute
\[ u_j = D_j b_j \] and \[ v_j = G_j b_j, \quad j = 1, 2, \ldots, n. \]

Compute
\[ u'_i = \sum_{j=1}^{i-1} u_j \] and \[ v'_i = \sum_{j=i}^{n} v_j, \quad i = 1, 2, \ldots, n. \]

Compute
\[ x_i = - E_i R^{-1} u'_i + F_i S^{-1} v'_i, \quad i = 1, 2, \ldots, n. \]

The recurrence relations of Algorithm 4.1 which define blocks \( D_i, E_i, F_i, G_i, R, S \) can be computed by means of the parallel algorithms presented in Section 3.

Algorithm 4.3 (Computation of the inverse).

Let \( U_1 = I_m, V_1 = V_2 = 0. \)

Compute
\[ U_k = - A_{k-1,k-1} A_{k,k-1}^{-1}, \quad k = 2, 3, \ldots, n. \]
\[ V_k = - A_{k-2,k-1} A_{k,k-1}^{-1}, \quad k = 3, 4, \ldots, n. \]

Compute
\[ D_k = Q_{kk}, \quad k = 2, 3, \ldots, n, \quad \text{with one of the algorithms of Section 3}, \]
\[ R = -(D_{n-1}A_{n-1,n} + D_n A_{n,n}). \]

Let \( U_1 = I_m, V_1 = V_2 = 0. \)

Compute
\[ U_k = - A_{n-k+2,n-k+2}^{-1} A_{n-k+2,n-k+2}, \quad k = 2, 3, \ldots, n. \]
\[ V_k = - A_{n-k+2,n-k+2}^{-1} A_{n-k+2,n-k+3}, \quad k = 3, 4, \ldots, n. \]
Compute

\[ E_{n-k+1} = P_{kk}, \quad k = 2, 3, \ldots, n, \] with one of the algorithms of Section 3.

Let \( U_1 = I_m, \ V_1 = V_2 = 0. \)
Compute

\[ U_k = -A_{k-1,k}^{-1}A_{k-1,k-1}, \quad k = 2, 3, \ldots, n, \]
\[ V_k = -A_{k-1,k}^{-1}A_{k-1,k-2}, \quad k = 3, 4, \ldots, n. \]

Compute

\[ F_k = P_{kk}, \quad k = 2, 3, \ldots, n, \] with one of the algorithms of Section 3,
\[ S = -(A_{n-1,n}F_{n-1} + A_{n,n}F_n). \]

Let \( U_1 = I_m, \ V_1 = V_2 = 0. \)
Compute

\[ U_k = -A_{n-k+2,n-k+2}^{-1}A_{n-k+1,n-k+2}, \quad k = 2, 3, \ldots, n. \]
\[ V_k = -A_{n-k+3,n-k+2}^{-1}A_{n-k+1,n-k+2}, \quad k = 3, 4, \ldots, n. \]

Compute

\[ G_{n-k+1} = Q_{kk}, \quad k = 2, 3, \ldots, n, \] with one of the algorithms of Section 3.

It is worth noting that Algorithms 4.1, 4.2, and 4.3 consist of recurrence relations analogous to the ones used by Stone. Therefore the question of their stability can be addressed in terms of the discussion presented in [14, 15].

5. IMPLEMENTATION ON A SMALL NUMBER OF PROCESSORS

Algorithms 4.1 and 4.2 can be easily implemented on a small number of processors (e.g. 1, 2, 4, or 8). Let \( \mathcal{C}, \ \mathcal{M}, \) and \( \mathcal{A} \) denote the time complexity, on the available processors, of \( m \times m \) matrix inversion, multiplication, and addition, respectively; moreover let \( \mathcal{M} \) and \( \mathcal{A} \) denote the complexity of \( m \times m \) matrix-vector multiplication and \( m \)-vector addition, respectively. Fi-
Algorithm 4.1 consists of four groups of equations which can be computed concurrently, so that the complexity bounds presented in the first three rows of Table 3 can be easily derived. An alternative implementation on 4 processors consists of computing first the inverses of the off-diagonal blocks of \( T \) on 4 processors, and then communicating the results at the expense of \( 2n - 2 \) communications. Using 8 processors it is possible to perform concurrently (1) the inversion of the off-diagonal blocks and (2) two block multiplications and one block addition. The parallel complexity depends on the relative costs of \( \mathcal{E} \) and \( 2\mathcal{M} + \mathcal{A} \).

Algorithm 4.2 consists of four steps; the parallelization on two processors is trivial, and \( n \) communications suffice in order to perform the additions in the fourth step. Using more than two processors, the \( 6n \) multiplications of steps 2 and 4 can be easily parallelized, the additions of step 3 can be

---

**TABLE 3**

**COMPLEXITY OF THE IMPLEMENTATION OF THE INVERSION ALGORITHM ON A SMALL NUMBER OF PROCESSORS**

<table>
<thead>
<tr>
<th>No. of processors</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((12n - 16)\mathcal{M} + (4n - 6)\mathcal{A} + (2n - 2)\mathcal{E})</td>
</tr>
<tr>
<td>2</td>
<td>((6n - 8)\mathcal{M} + (2n - 3)\mathcal{A} + (n - 1)\mathcal{E})</td>
</tr>
<tr>
<td>4</td>
<td>((3n - 3)\mathcal{M} + (n - 2)\mathcal{A} + (n - 1)\mathcal{E}), ((3n - 3)\mathcal{M} + (n - 2)\mathcal{A} + [(n - 1)/2]\mathcal{E} + (2n - 2)\mathcal{C})</td>
</tr>
<tr>
<td>8</td>
<td>((n - 1)\mathcal{M} + (n - 1)\mathcal{A} + (n - 1)\mathcal{E}) if ( \mathcal{E} \geq 2\mathcal{M} + \mathcal{A} ), ((2n - 2)\mathcal{M} + (n - 2)\mathcal{A} + (n - 1)\mathcal{E}) if ( \mathcal{E} &lt; 2\mathcal{M} + \mathcal{A} )</td>
</tr>
</tbody>
</table>

---

**TABLE 4**

**COMPLEXITY OF THE IMPLEMENTATION OF THE SOLVING ALGORITHM ON A SMALL NUMBER OF PROCESSORS**

<table>
<thead>
<tr>
<th>No. of processors</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(8nm + 3na + 2\mathcal{E})</td>
</tr>
<tr>
<td>2</td>
<td>(3nm + 2na + \mathcal{E} + nc)</td>
</tr>
<tr>
<td>4</td>
<td>([\lfloor n/2 \rfloor]m + 2na + \mathcal{E} + ([n/2] + 1)c)</td>
</tr>
<tr>
<td>8</td>
<td>([\lfloor n/4 \rfloor]m + 2na + \mathcal{E} + ([n/4] + 3)c)</td>
</tr>
</tbody>
</table>
performed on different processors although not concurrently, and the number of communications decreases as the number of processors increases. The complexity of Algorithms 4.1 and 4.2 implemented on systems with 1, 2, 4, or 8 processors is presented in Tables 3 and 4, respectively.

6. IMPLEMENTATION ON A LARGE NUMBER OF PROCESSORS

A parallel architecture well suited to implement our algorithms for computing the inverse of a block tridiagonal matrix and/or solving the associated linear system is the hypercube. This architecture belongs to a class of machines widely studied in the literature (e.g. see [9, 12, 13]) and consists of \( n = 2^h \) processors, each connected to \( h \) neighbors. Each processor is identified by an \( n \)-bit binary word, and it is connected to the \( h \) processors, whose words can be derived by altering one digit.

Let \( d[i, j] \) be the length of a minimal path on the hypercube from processor \( i \) to processor \( j \). It readily follows that

\[
d[i, j] = \text{number of bits differing in the words associated to } i \text{ and } j.
\]

In our implementation, an integer address in the range \([0, n - 1]\) is associated to each binary word according to a particular Gray code, namely the inverted binary code [4, p. 16]. The peculiar property of Gray codes is that the words associated to numbers \( k \) and \( k - 1 \) differ exactly by one bit, i.e.

\[
d[k, k - 1 \mod n] = 1.
\]

Moreover for inverted binary code it is straightforward to prove that

\[
d[i, i + 2^j \mod n] = 2, \quad 0 < j < h.
\]

Algorithms 3.1 and 3.2 can be implemented on the hypercube by storing \( U_k \) and \( V_k \) in the processor \( k \) and by performing the computations of \( P_{ik}, Q_{ik} \), for the required values of \( i \), using the same processor. It is easy to see that communications are needed between processors whose addresses differ by a number which is the sum of two powers of two. Then it readily follows that in any equation the communication cost is at most 4. Therefore the resulting time complexity is \( O(S \log n) \) for both algorithms.
A more detailed analysis allows proving that the time complexity on the hypercube of both Algorithm 3.1 and Algorithm 3.2 is

\[(9.\mathcal{M} + 3.\mathcal{A} + 30.\mathcal{C}) \log n + O(1).\]

Using either Algorithm 3.1 or Algorithm 3.2 for the implementation of Algorithm 4.3 on the hypercube, with a suitable placement of problem data, it is easy to see that the computation of \(U_k\) and \(V_k\) for each of the four parts of the algorithm and the computation of \(R\) and \(S\) can be performed with cost not depending on \(n\). The complexity of Algorithm 4.3 then becomes

\[(36.\mathcal{M} + 12.\mathcal{A} + 30.\mathcal{C}) \log n + O(1).\]

To evaluate the complexity of Algorithm 4.2, it is useful to study the complexity of the computation on the hypercube of the quantities

\[y_i = \sum_{j=1}^{i-1} z_j, \quad i = 1, 2, \ldots, n.\]

**Algorithm 6.1.** The hypercube of size \(n\) (say \(H_n\)) is recursively divided into two equal hypercubes \((H'_n/2\) and \(H''_n/2\)). Processor \(i\) of \(H'_n/2\) computes the two quantities

\[y'_i = \sum_{j=1}^{i-1} z_j \quad \text{and} \quad y'_n/2 = \sum_{j=1}^{n/2} z_j.\]

Analogously processor \(n/2 + i\) of \(H''_n/2\) computes the two quantities

\[y''_{n/2+i} = \sum_{j=n/2+1}^{n/2+i-1} z_j \quad \text{and} \quad y''_n = \sum_{j=n/2+1}^{n} z_j.\]

The combination step consists of computing the two quantities

\[y_i = y'_i \quad \text{and} \quad y_n = y'_n/2 + y''_n\]

on processors \(i, i = 1, 2, \ldots, n/2,\) and

\[y_{n/2+i} = y'_n/2 + y''_{n/2+i} \quad \text{and} \quad y_n = y'_n/2 + y''_n.\]
on processors $n/2 + i$, $i = 1, 2, \ldots, n/2$. The first step of the recursion consists of computing on two adjacent processors the sum of their contents. The resulting complexity can be easily derived from the relations

\[
T(n) = T(n/2) + 2a + 2\varepsilon,
\]

\[
T(2) = a + 2\varepsilon,
\]

i.e.,

\[
T(n) = (2a + 2\varepsilon)(\log n - 1) - a.
\]

Therefore, Algorithm 4.2 can be executed on the hypercube in time

\[ (2\varepsilon + 4a + 4\varepsilon) \log n + O(1). \]

The hypercube implementation is possible even if the number $q = 2^h$ of processors is less than $n$. Assume $n = 2^p$ ($p > h$). By grouping at each processor the computations of $2^{p-h}$ nodes, the performance of the system is slowed by a factor $n/q$, and the communication costs are still $O(\log q)$.

Moreover, if the hypercube consists of $n$ sequential processors, then $\mathcal{P} = O(m^3)$ and the time complexity for solving a block tridiagonal linear system results to be $O(m^3 \log n)$.

Siegel [12, 13] has analyzed various interconnection networks and the complexity of simulating each network with each of the other ones. In particular he has shown that the perfect shuffle can emulate the hypercube with an $O(\log n)$ factor of loss due to increased communication costs. This leads to a time complexity $O(m^3 \log^2 n)$ for solving a block tridiagonal linear system on a perfect shuffle consisting of $O(n)$ sequential processors.

This result can be compared with the one presented in [8]. The order of complexity attained in [8] for solving a block tridiagonal linear system is $O(m^{5/2}n^{1/2})$ by using $(m/n)^{1/2}$ processors and a perfect shuffle interconnection network.

REFERENCES

3 L. Euler, *Introductio in Analysin Infinitorum*, Lausanne, 1748, Section 259.

Received 15 September 1986; final manuscript accepted 22 July 1987