ACKNOWLEDGMENTS

The authors would like to thank Reviewer G for improving the article, and also Prof. I.F. Blake and M.Z. Wang for their useful comments during the manuscript preparation.

This work was supported, in part, by the University of Waterloo under a start-up grant, and, in part, by the Canadian Institute for Telecommunications Research under the NCE program. A preliminary version of part of this article was presented at the 16th Biennial Symposium on Communications, Kingston, Ontario, Canada. [1]

REFERENCES


Abstract—Based on the parallel minimal norm method an algorithm is derived to solve tridiagonal linear systems with a high degree of parallelism. No conditions need to be posed with respect to the system. Experiments indicate that the numerical stability of the algorithm is similar to Gaussian elimination with partial pivoting.

Index Terms—Parallel algorithms, parallel minimal norm method, tridiagonal linear systems, row-oriented orthogonalization, structural orthogonality.

I. INTRODUCTION

Large tridiagonal linear systems arise in many fields of numerical computation. New algorithms for these systems have been developed that take advantage of the architecture of parallel computers. Ortega [9] gives an overview. Parallel algorithms include Cyclic Reduction [8], VLSI processors [9], Recursive Doubling [10] and the methods of Sun [11] and Bondell [2]. At this moment parallel machines are available that consist of many thousands of processors [6], [12]. As in the near future the number of processors will increase even more, the degree of parallelism becomes a dominant property of algorithms. The algorithm presented here possesses a high degree of parallelism.

The parallel minimal norm method [3] is based on an inner product equation representation of a linear system. Each equation is an inner product of a known vector and the unknown solution that must satisfy a right-hand side. By orthogonalizing the inner product equations using the divide-and-conquer strategy [1].

II. ALGORITHM

A tridiagonal linear system in inner product notation looks like

\[
\begin{align*}
(a_1 & a_2 \ldots a_n) \\
\end{align*}

\[
\begin{pmatrix}
2 & \ldots & 0 \\
\vdots & \ddots & \vdots \\
0 & \ldots & 2
\end{pmatrix}

\begin{pmatrix}
1 \\
\vdots \\
1
\end{pmatrix}

= x_1 \\
\vdots \\
= b_n
\]

(1)

for \(a_i, \ldots, a_n, x, b \in \mathbb{R}^n\). The structural orthogonality of the vectors satisfies \(\langle a_i, a_j \rangle = 0\), for \(i, j \geq 3\). Therefore the system is divided into three sets of equations. Kamath [7] applies the same approach to introduce parallelism in block-tridiagonal linear systems. The size of the system is assumed to be \(n = 5 \times 2^m\) with \(m \in \mathbb{N}\). The initial set \(S\) consists of all \(n\) inner product equations, or \(S = \{1, 2, 3, \ldots, n\}\). The following division is applied.
Consequently the three sets consist of the following inner product equations.

\begin{align*}
S_1: \quad (a_1, x) &= b_1, \\
S_2: \quad (a_2, x) &= b_2, \\
S_3: \quad (a_3, x) &= b_3,
\end{align*}

(3)

The number of equations within each set equals \( n_i = \frac{n}{3} \) for \( i = 1, 2, 3 \). In order to be relieved from the absolute indices of the inner product equations, the symbolic notation is used. The index \( s_j \) refers to the \( j \)th inner product equation in set \( S_j \). In this way the notation can be kept simple.

The nonzero structures of the vectors of the inner product equations of the three sets are

\begin{align*}
S_1: \quad a_1 &= (*, *, *), \\
S_2: \quad a_2 &= (*, *, *), \\
S_3: \quad a_3 &= (*, *, *),
\end{align*}

(4)

Within each set the vectors of the inner product equations are orthogonal. However this is not the case for vectors of different sets. So orthogonalization is needed between the sets. Based on symmetry considerations the sets \( S_2 \) and \( S_3 \) are merged with respect to \( S_1 \).

Each vector in \( S_2 \) and \( S_3 \) only overlaps with the two neighboring vectors in \( S_1 \). Exceptions are the first vector in \( S_2 \) and the last vector in \( S_3 \) which overlap with only one vector in \( S_1 \). The following equations show the mergings of \( S_2 \) and \( S_3 \). The merging of \( S_2 \) is represented by \( \text{merge}(S_2, S_1) \) in a compact notation.

\begin{align*}
b_{n_2} := b_{n_2} - \frac{a_{n_2} \cdot a_{n_1}}{\| a_{n_1} \|} a_{n_1}, & \quad \text{for } j = 2, 3, \ldots, n; \\
b_{n_3} := b_{n_3} - \frac{a_{n_3} \cdot a_{n_1}}{\| a_{n_1} \|} a_{n_1} - \frac{a_{n_3} \cdot a_{n_2}}{\| a_{n_2} \|} a_{n_2} - \frac{a_{n_3} \cdot a_{n_3}}{\| a_{n_3} \|} a_{n_3}, & \quad \text{for } j = 2, 3, \ldots, n_3.
\end{align*}

(5, 6)

The resulting vectors of the inner product equations of \( S_2 \) and \( S_3 \) are now orthogonal to those of \( S_1 \). The nonzero patterns of the vectors of \( S_2 \) and \( S_3 \) are

\begin{align*}
S_2: \quad a_2^T &= (*, *, *), \\
S_3: \quad a_3^T &= (*, *, *),
\end{align*}

(7)

The average bandwidth of the vectors is doubled with respect to the original vectors. The structural properties of the vectors of both sets satisfy \( (a_j, a_j) = 0 \), for \( j = 1 \). Therefore for both sets a division into two sets becomes obvious. The set \( S_2 \) is divided into a new \( S_2 \) and a new \( S_1 \). The indices \( s_j \) with \( j \) even are contained in the new \( S_1 \). The indices \( s_j \) with \( j \) odd move to \( S_2 \). For set \( S_3 \) the division is reverse. The new \( S_3 \) contains the indices \( s_j \) with \( j \) odd and the new \( S_4 \) contains the indices \( s_j \) with \( j \) even.

\begin{align*}
S_2: \quad (a_2, x) &= b_2, \\
S_3: \quad (a_3, x) &= b_3, \\
S_4: \quad (a_4, x) &= b_4,
\end{align*}

(8)

After the divisions the number of equations in \( S_2, S_3, S_4 \) is halved, it satisfies \( n_i = \frac{n}{4} \) for \( i = 2, 3, 4, 5 \). The nonzero patterns of the vectors of the new \( S_2 \) and \( S_3 \) are identical.
The nonzero patterns of the vectors of $S_4$ and $S_5$ look like

$$S_4: \begin{align*}
S_{4i}^T &= (\ast, \ast, \ast, \ast, \ast), \\
S_{4i/e}^T &= (\ast, \ast, \ast, \ast, \ast), \\
S_{4i/e+1}^T &= (\ast, \ast, \ast, \ast, \ast). 
\end{align*}$$

$$S_5: \begin{align*}
S_{5i}^T &= (\ast, \ast, \ast, \ast, \ast), \\
S_{5i/2}^T &= (\ast, \ast, \ast, \ast, \ast), \\
S_{5i/2+1}^T &= (\ast, \ast, \ast, \ast, \ast). 
\end{align*}$$

The vectors within $S_2$ are orthogonal. This holds for $S_3$ as well. However corresponding vectors in $S_4$ and $S_5$ are not orthogonal. Therefore a merger has to be done. The merging of $S_4$ with respect to $S_2$ involves only one orthogonalization per inner product equation. The nonzero pattern of $S_5$ does not change. In compact notation this merger is $\text{merge}(S_4S_5)$.

$$b_{4i} := b_{4i} - \frac{b_{4i} \cdot b_{5i}}{\|b_{5i}\|^2} b_{5i}, \text{ for } j = 1, 2, \ldots, n_i; \tag{18}$$

$$b_{4i} := b_{4i} - \frac{b_{4i} \cdot b_{5i}}{\|b_{5i}\|^2} b_{5i}, \text{ for } j = 1, 2, \ldots, n_i. \tag{19}$$

Again there exists symmetry between the nonzero patterns of $S_4$ and $S_5$ and those of $S_2$ and $S_3$. Therefore a similar procedure can be followed as with $\text{merge}(S_4S_5)$ and $\text{merge}(S_3S_2)$. The only difference is that the mergings of $S_4$ and $S_5$ are not with respect to one but with respect to two sets. In compact notation the merger of $S_4$ is $\text{merge}(S_4S_3)$. In compact notation the merger of $S_5$ is $\text{merge}(S_5S_3)$.

$$a_{5j} := a_{5j} - \sum_{i=1}^{n_j} \frac{b_{5j} \cdot b_{5i}}{\|b_{5i}\|^2} b_{5i}, \text{ for } j = 1, 2, \ldots, n_j; \tag{20}$$

$$b_{5j} := b_{5j} - \sum_{i=1}^{n_j} \frac{b_{5j} \cdot b_{5i}}{\|b_{5i}\|^2} b_{5i}, \text{ for } j = 1, 2, \ldots, n_j; \tag{21}$$

$$b_{5j} := b_{5j} - \sum_{i=1}^{n_j} \frac{b_{5j} \cdot b_{5i}}{\|b_{5i}\|^2} b_{5i}, \text{ for } j = 1, 2, \ldots, n_j. \tag{22}$$

Therefore the previous procedure of the divisions and the subsequent mergences is generalized. In compact notation each step is given by

$$\text{divide } S_2 \rightarrow S_2, S_{2r+1}; \text{ divide } S_{2r+1} \rightarrow S_{2r+1}, S_{2r+2}; \text{ divide } S_{2r+2} \rightarrow S_{2r+2}, S_{2r+3};$$

$$\text{merge } (S_4, S_5); \text{ merge } (S_2, S_{2r+1}); \text{ merge } (S_2, S_{2r+2}); \text{ merge } (S_2, S_{2r+3}).$$

The procedure has to be applied $\log_2(n)$ times. The assumption about the size of the system $n$ was made in order to maintain the active sets at equal size during each stage of the computation. The total number of sets is $n = 2^{\log_2(n)} + 3$. The last two sets are orthogonal to all other sets. Both sets consist of only one equation with a full vector. The vectors are not yet orthogonal, therefore the last set is merged with respect to the last but one set (in compact notation $\text{merge}(S_3, S_2)$).

To each orthogonal set belongs a minimal norm solution which is the sum of the minimal norm solutions of its orthogonal inner product equations.

$$x_i = \sum_{j=1}^{n_j} b_{ij} / \|b_{ij}\|, \text{ for } i = 1, 2, \ldots, n_i. \tag{30}$$

This is referred to as $\text{solve}(S)$ in compact notation. Finally the solu-
tion of the complete system, i.e., (1), is found as the sum of the
minimal norm solutions of all sets.

\[ x = \sum_{i=1}^{n} \beta_i. \]  (31)

In Fig. 1a the algorithm is presented in compact notation.

\[
\text{init: divide } \mathcal{S} \rightarrow \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4 \\
\text{merge(}\mathcal{S}_1(\mathcal{S}_2)\text{); merge(}\mathcal{S}_3(\mathcal{S}_4)\text{)} \\
\text{loop: } i = 1, 2, \ldots, \text{loop(3)} \\
\quad \text{divide } \mathcal{S}_4 \rightarrow \mathcal{S}_{4a}, \mathcal{S}_{4b} \\
\quad \text{divide } \mathcal{S}_{4a} \rightarrow \mathcal{S}_{4a1}, \mathcal{S}_{4a2} \\
\quad \text{merge(}\mathcal{S}_{4a1}(\mathcal{S}_{4a2})\text{)} \\
\quad \text{merge(}\mathcal{S}_{4a3}(\mathcal{S}_{4a4})\text{)} \\
\text{merge(}\mathcal{S}_{4b}(\mathcal{S}_{4c})\text{)} \\
\quad \text{solve(}\mathcal{S}_{1}\text{); solve(}\mathcal{S}_{2}\text{)} \\
\quad \text{solve(}\mathcal{S} \text{)} \\
\text{end: merge(}\mathcal{S}_{1}(\mathcal{S}_{2}\ldots \mathcal{S}_{n-1})\text{)} \\
\text{solve(}\mathcal{S}_{1}\text{); solve(}\mathcal{S}_{2}\text{)} \\
\text{return } x = x_1 + \cdots + x_n.
\]

Fig. 1a. Standard algorithm.

\[
\text{init: divide } \mathcal{S} \rightarrow \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4 \\
\text{merge(}\mathcal{S}_1(\mathcal{S}_2)\text{); merge(}\mathcal{S}_3(\mathcal{S}_4)\text{)} \\
\text{solve(}\mathcal{S}_1\text{)} \\
\text{return } x = x_1 \\
\text{loop: } i = 1, 2, \ldots, \text{loop(3)} \\
\quad \text{divide } \mathcal{S}_3 \rightarrow \mathcal{S}_{3a}, \mathcal{S}_{3b} \\
\quad \text{divide } \mathcal{S}_{3a} \rightarrow \mathcal{S}_{3a1}, \mathcal{S}_{3a2} \\
\quad \text{merge(}\mathcal{S}_{3a1}(\mathcal{S}_{3a2})\text{)} \\
\quad \text{merge(}\mathcal{S}_{3a3}(\mathcal{S}_{3a4})\text{)} \\
\text{merge(}\mathcal{S}_{3b}(\mathcal{S}_{3c})\text{)} \\
\quad \text{solve(}\mathcal{S}_1\text{); solve(}\mathcal{S}_2\text{)} \\
\quad \text{solve(}\mathcal{S}_4\text{)} \\
\text{end: merge(}\mathcal{S}_{1}(\mathcal{S}_{2}\ldots \mathcal{S}_{n-1})\text{)} \\
\text{solve(}\mathcal{S}_{1}\text{); solve(}\mathcal{S}_{2}\text{)} \\
\text{return } x = x + x_{n-1} + \cdots + x_n.
\]

Fig. 1b. Modified algorithm.

### III. MEMORY REQUIREMENTS

Within an orthogonal set the vectors do not overlap. The nonzero
patterns of the consecutive vectors exactly succeed each other. One n
element register can be used to store all vectors of a set. Conse-
quently 2log₂(\( \frac{n}{3} \)) + 3 registers are needed for the complete algo-

A simple modification of the algorithm (Fig. 1b) significantly re-
duces the memory requirements. At the start of each stage consecu-
tive vectors in \( \mathcal{S}_2 \) and \( \mathcal{S}_{4a} \) overlap with half the bandwidth. After the
divisions there exist four sets where the consecutive vectors within
each set do not overlap. Thus four n element registers are needed for

The four sets. The \( \text{merge(}\mathcal{S}_{4a}(\mathcal{S}_2)\text{)} \) does not change the nonzero
pattern of \( \mathcal{S}_{4a} \), thus no additional storage is required. The
\( \text{merge(}\mathcal{S}_{4a}(\mathcal{S}_2\ldots \mathcal{S}_{4c})\text{)} \) almost double the
nonzero elements in the vectors of \( \mathcal{S}_{4a}\ldots \mathcal{S}_{4c} \). Since these vectors
were stored consecutively, no vector can be overwritten without de-
stroying data of other vectors in the same register that have not been
processed yet. Therefore the new vectors of both sets must be stored
in four new n element registers. Since \( \mathcal{S}_2 \) and \( \mathcal{S}_{4a} \) are now orthogo-

IV. OPERATION COUNT

Only floating point operations contribute to the operation count.
In the modified algorithm only the \text{merge} and \text{solve} parts and the
summation of the minimal norm solutions of the sets involve floating
point operations. The \text{divide} part consists of indexed memory refer-
cences only. Each \text{merge} part consists of orthogonalizations of one
inner product equation with respect to another. An inner product
orthogonalization is given by

\[
\mathbf{b}_i := \frac{\mathbf{a}_i \cdot \mathbf{b}_j}{\| \mathbf{b}_j \|^2} \mathbf{b}_j.
\]

Suppose that \( \mathbf{a}_i \) and \( \mathbf{b}_j \) consist of p respectively q nonzero elements
and their overlap is equal to r nonzero elements. If the number of
floating point operations of an orthogonalization is represented by
\text{ort}(q, r), it satisfies \text{ort}(q, r) = 3(q + r) + 1. The \text{solve} part consists of
the computation of the minimal norm solution of an orthogonal set.
The minimal norm solution of the jth inner product equation of the set \( \mathcal{S}_j \) is

\[
\mathbf{x}_j = \frac{\mathbf{b}_j}{\| \mathbf{a}_j \|^2} \mathbf{b}_j.
\]

If \( \mathbf{a}_j \) consists of p nonzero elements, the computation requires 3p
floating point operations. Since there is no overlap between the vec-
tors within a set, the computation of the minimal norm solution of all
inner product equations within the set requires \( n \cdot 3 \cdot (\frac{p}{3}) = 3n \) flops.
In Table I the operation counts are shown. Summation of all contri-
butions gives the operation count of the modified algorithm.

\[
(56n - 4) \log_2 \left( \frac{n}{3} \right) - \frac{8n}{3} + 92.
\]

V. PARALLEL PROCESSING

The amount of parallelism within each \text{merge(}\mathcal{S}\text{)} is proportional
to the number of inner product equations that belong to both sets. As
the size of a set is halved after a division—the consecutive set sizes
946

when the number of processors is not too large. The communication overheads for hypercube architectures and they are only efficient.

Table I: Operation Count of the Different Parts of the Modified Algorithm

<table>
<thead>
<tr>
<th>Part</th>
<th>Number of Floating point operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>divide $S \to s_1, s_2, s_3$</td>
<td>$2(</td>
</tr>
<tr>
<td>$X = X_1$</td>
<td>$2n$</td>
</tr>
<tr>
<td>divide $S_0 \to S_{01}, S_{02}$</td>
<td>$\sqrt{2}\tau(2, 3, 3)$</td>
</tr>
<tr>
<td>divide $S_{01}, S_{02}$</td>
<td>$\sqrt{2}\tau(2, 3, 3)$</td>
</tr>
<tr>
<td>merge($S_{01}, S_{02}$)</td>
<td>$\sqrt{2}\tau(2, 3, 3)$</td>
</tr>
<tr>
<td>solve($S_0$)</td>
<td>$5n$</td>
</tr>
<tr>
<td>$X = X + X_0 + X_{01}$</td>
<td>$2n$</td>
</tr>
<tr>
<td>merge($S_n, S_{n-1}$)</td>
<td>$\tau(n, n)$</td>
</tr>
<tr>
<td>solve($S_{n-1}$)</td>
<td>$5n$</td>
</tr>
<tr>
<td>$X = X + X_{n-1} + X_n$</td>
<td>$2n$</td>
</tr>
</tbody>
</table>

are $\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \ldots, 1$ —the degree of parallelism decreases during the computation. However, the nonzero patterns of the vectors of consecutive pairs of sets double in size. The consecutive vector sizes are $3, 6, 12, \ldots, \frac{1}{2}, n$. Because each inner product equation orthogonalization consists of dot, square norm and saxpy computations, parallelization is possible as well here. The saxpy computation can be performed in parallel without any interaction. The dot and square norm computations require interaction when they are performed in parallel. The influence on performance and speedup highly depends on the architectural features of the parallel computer such as interconnection network, communication latency, vector size and the degree of parallelism increases as well. The total degree of the parallelism is the product of both components; it roughly remains constant during computation and it approaches $O(n)$. Wang's method has a smaller total operation count. The parallel time complexity is $O(\log n)$.

VI. RESULTS AND CONCLUSIONS

The modified algorithm was implemented in FORTRAN. It was run on the Convex C3820 and the CRAY Y-MP/4404. Triangular systems with $(a_0, a_1, a_1, a_1)$ equal to $(1, -2, 1, (1, -2, 1, 2, 1, 2, 1, 2), (1, -2, 1, 2, 1, 2, 1, 2), (1, 0, 0, 1, 0) \text{ and } (1, 2, 3, 4)^T$ were tested. Note that several of them do not belong to the symmetric, positive definite or diagonal dominant class. The error was compared with the error of Gaussian elimination with partial pivoting. No significant differences were observed. These measurements indicate a favorable error behavior.

Table II: Characteristics of Several Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Operation count</th>
<th>Av. DOP</th>
<th>Parallel time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Elimination</td>
<td>$8n$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Cyclic Reduction</td>
<td>$7n$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Recursive Doubling</td>
<td>$7n \log n$</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Parallel Minimal Norm</td>
<td>$5n \log n$</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Wang's method ($p = \sqrt{n}$)</td>
<td>$2n + 12n - 8$</td>
<td>$O(\sqrt{n})$</td>
<td>$O(\sqrt{n})$</td>
</tr>
</tbody>
</table>

overheads were not investigated for the parallel minimal norm method. Although the algorithm is costly in absolute terms in comparison with other algorithms, the degree of interaction is limited and the communication overheads should be small. Therefore the algorithm should compete well with other methods.

All existing parallel algorithms impose restrictions on the kind of linear systems to be solved, such as symmetry, positive definiteness and/or diagonal dominance to guarantee numerical stability. For general tridiagonal systems pivoting is required, which destroys the parallel properties of the algorithms. The experiments indicate that no such requirements are needed for the parallel minimal norm method. This robustness will be advantageous in a parallel environment. Further research has to be done, especially implementations on local memory architectures with a large number of processors (MPPs) will be of interest.

ACKNOWLEDGMENTS

The authors would like to thank the Stichting Nationale Computer Faciliteten for the grant that was awarded to perform experiments on the CRAY Y-MP/4404.

REFERENCES