AN M-STEP PRECONDITIONED CONJUGATE GRADIENT METHOD FOR PARALLEL COMPUTATION

Loyce Adams

Contract Nos. NAS1-15810, NAS1-17070, NAS1-17130
June 1983

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

NASA Contractor Report 172150

NASA-CR-172150
19830020669
AN M-STEP PRECONDITIONED CONJUGATE GRADIENT METHOD FOR PARALLEL COMPUTATION

Loyce Adams
Institute for Computer Applications in Science and Engineering
Hampton, Virginia 23665

Abstract — This paper describes a preconditioned conjugate gradient method that can be effectively implemented on both vector machines and parallel arrays to solve sparse symmetric and positive definite systems of linear equations. The implementation on the CYBER 203/205 and on the Finite Element Machine is discussed and results obtained using the method on these machines are given.

Introduction

In this paper we are concerned with the solution of a sparse \( N \times N \) system of symmetric and positive definite linear equations

\[ Ku = f \tag{1.1} \]

by preconditioned conjugate gradient (PCG) methods on both vector computers and parallel arrays. Several descriptions of these methods appear in the literature; see for example, Concus, Golub, O'Leary [1976] and Chandra [1978]. Also, Schreiber [1978] discussed the implementation of conjugate gradient (CG) on vector computers and Podsiadlo and Jordan [1981] discussed its implementation on the Finite Element Machine under construction at NASA Langley Research Center.

The PCG method solves the system \( Ku = f \) where

\[ K = Q^{-\frac{1}{2}}HK^{-1}Q^{-\frac{1}{2}}, \quad u = Q^{-1}r, \quad f = Q^{-1}f, \tag{1.2} \]

\( Q \) is a nonsingular matrix, and the symmetric and positive definite preconditioning matrix is given by \( K = QQ' \). The algorithm for the solution of \( u \) directly is described in Chandra [1978] and is given below where \( u, r, f \), and \( p \) are vectors and \( (x,y) \) denotes the inner product \( x'y \).

1. Choose \( u^0 \)
2. \( r^0 = f - Ku^0 \)
3. \( Hr^0 = r^0 \)
4. \( r^0 \)
5. For \( k = 0,1,\ldots \)
   \[ \alpha = \frac{(r^k', r^{k+1})}{(p^k, Kp^k)} \]
   \[ u^{k+1} = u^k + \alpha p^k \]
   \[ r^{k+1} = r^k - \alpha p^k \]
6. If \( |r^{k+1} - r^k| < \epsilon \) then stop, otherwise continue.
7. \( p_{k+1} = p^k + \beta p^k \)

Algorithm 1. PCG Algorithm

We note that the standard conjugate gradient algorithm results by choosing \( H = I \).

For vector machines, if \( H = I \), all steps of the iteration loop except (1) and (6) can be vectorized. In particular, the multiplication \( Kp \), for \( K \) sparse, vectorizes after a suitable ordering of the equations and will be discussed in detail in Section 3. The difficulty arises in the formation of the inner products necessary to calculate \( \alpha \) and \( \beta \). These calculations require a phase in which \( N \) partial sums must be added together and therefore do not vectorize well.

For parallel arrays like the Finite Element Machine (Jordan [1978], Adams [1982]), the calculation of \( u, r, f \), and \( p \) can be distributed to the individual processors and the necessary communication between processors can be performed on the dedicated local links. The convergence test in (3) can be performed by using flag network. However, for a large number of processors, the calculations of \( \alpha \) and \( \beta \) can be expensive since the number of values to be summed for each inner product is equal to \( P \), the number of processors. Jordan [1979] realized that this was potentially detrimental to the efficiency of the method on this machine, and as a result, a special hardware circuit (sum/max) was designed to perform the \( P \) sums in \( O(\log P) \) time.

Since Algorithm 1 has two inner products per iteration that will become costly as \( N \) (on vector machines) or \( P \) (on arrays) increases, a natural goal is to devise a preconditioner that will reduce the number of CG iterations, and hence the number of inner products, while being inexpensive to implement. In the next section preconditioners that are based on taking \( m \) steps of an iterative method are described. In Section 3, the implementations of these methods on the CYBER 203/205 and the Finite Element Machine are...
given for a system of equations that results from an example structural engineering problem. Results for this problem on the CYBER 205 and the Finite Element Machine are given in Section 4.

2. M-Step Parallel Preconditioners

2.1 Choosing H

The preconditioned conjugate gradient algorithm of the last section requires a symmetric and positive definite preconditioning matrix H. The question is how to choose H so that the condition number of K

\[ \kappa(K) = \frac{\kappa(HK)}{\kappa(H)} \]

is as small as possible.

The best choice for H in the sense of minimizing \( \kappa(K) \) is \( H = K \) but this gains nothing since \( K r = r \) is just as difficult to solve as \( K u = f \). A class of preconditioners that appears to be easily implemented on parallel computers arises by choosing H to be a splitting of K that describes a linear stationary iterative method. As an example, the SSOR splitting of K yields

\[ H = \frac{\omega}{\omega-1} (D-L)^{-1}(D-U) \]  \hspace{1cm} (2.1)

where \( D-L \) and \( -U \) are the diagonal, strictly lower, and strictly upper parts of \( K \) respectively. This splitting has been considered extensively in the literature as a preconditioner; for example, refer to Concus, Golub, O'Leary [1976] and the references therein. Now, if the matrix K is ordered by the Multicolor ordering (Adams and Ortega [1982]), the system \( Hr = r \) can be implemented on parallel computers as a forward followed by a backward Multicolor SOR iteration applied to \( Kr = r \) with initial guess \( \overline{z}^{(0)} = 0 \) and will be explained in more detail in Section 3. The question now arises whether it would be beneficial to take more than one step of a linear stationary iterative method to produce a preconditioner H that more closely approximates K. If this is done, the resulting preconditioning matrix is

\[ H = P(I+G+\ldots+G^{m-1})^{-1} \]  \hspace{1cm} (2.2)

Now, H must be symmetric and positive definite to be considered as a preconditioner. The necessary and sufficient conditions for H to satisfy these requirements are given in Adams [1982] and we only note here that \( [I] = \omega I \) in the SSOR splitting matrix these conditions are met. We also note that Dubois, Greenbaum, and Rodrique [1979] considered a truncated Neumann series for \( K^{-1} \) as a preconditioner which corresponds to a Jacobi splitting where \( P = \text{diag}(K) \).

Even though the preconditioner in (2.2) for the Jacobi splitting is symmetric and positive definite, the question of how well the resulting PCG method will reduce the number of CG iterations must be answered. In Adams [1982], for the SSOR splitting, the condition number of the matrix K of (1.2) was proven to decrease as the number of steps of the preconditioner in (2.2) increases; however, the maximum ratio of \( \kappa(K) \) was shown to be m. In practice, for larger m, this reduction may not be enough to balance the increase in the work that must be done by the preconditioner (as results in Section 4 verify). However, by parametrizing this preconditioner, the method is very effective. This parametrization is briefly discussed in the next section and the parameters for the SSOR splitting are given.

2.2 Parametrizing H

Johnson, Hiebich, and Paul [1982] have suggested symmetrically scaling the matrix K to have unit diagonal and then take m terms of a parametrized Neumann series for \( K^{-1} = (I - G)^{-1} \) as the value for \( H^{-1} \). This corresponds to a symmetric preconditioning matrix whose inverse is a polynomial of degree \( m-1 \) in G,

\[ H^{-1} = a_0 I + a_1 G + a_2 G^2 + \ldots + a_{m-1} G^{m-1} \]  \hspace{1cm} (2.3)

derived from the Jacobi splitting.

\[ K = I - G \]  \hspace{1cm} (2.4)

of K; hence, the solution to \( H r = r \) can be implemented by taking m steps of the Jacobi iterative method applied to \( Kr = r \) with initial guess \( \overline{z}^{(0)} = 0 \). Johnson, et al., choose \( a_i \)'s so that the eigenvalues of \( H^{-1} K \) are positive on the interval \( [\lambda_1, \lambda_m] \) that contains the eigenvalues of K and hence are close to 1 as possible in some sense such as the min-max or the least squares criteria. Clearly, if \( m = 1 \), \( H^{-1} K = \alpha_0 K \) and the condition number of \( H^{-1} K \) is the same for all \( a_0 \neq 0 \). Hence, we are only interested in \( m > 1 \).

We now generalize this idea for any splitting of the matrix K,

\[ K = P = Q \]  \hspace{1cm} (2.5)

If \( G = F^{-1} Q \), then by parametrizing (2.2), the inverse of the m-step preconditioner becomes

\[ H^{-1} = (a_0 I + a_1 G + a_2 G^2 + \ldots + a_{m-1} G^{m-1})^{-1} \]  \hspace{1cm} (2.6)

and will be symmetric if \( P \) is symmetric. We choose the values of \( a_i \) so that the eigenvalues of \( H^{-1} K \) are positive on the interval \( [\lambda_1, \lambda_m] \) that contains the eigenvalues of \( F^{-1} K \) and are close to 1 as possible in some sense such as the min-max or least squares criteria. For the least squares criteria, the values of \( a_i \) that correspond to the SSOR splitting are given in Table 1 for \( m = 2, 3, \) and 4.

<table>
<thead>
<tr>
<th>( m )</th>
<th>( a_0 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.00</td>
<td>5.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>-2.00</td>
<td>7.00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>7.00</td>
<td>-24.50</td>
<td>31.50</td>
</tr>
</tbody>
</table>

In the next section we describe how to implement the m-step parametrized SSOR PCG method on the
3. Implementation of the m-step SSOR PCG Method

We first describe the algorithm for solving $Hr = r$, where $H$ is the preconditioning matrix given by (2.6). To be concrete, this description will be given for the following test problem.

The domain considered will be a rectangular plate discretized with triangular finite elements over which linear basis functions are defined. The nodes of the triangles are colored Red, Black, and Green so that nodes on a given triangle are different colors as shown in Figure 1. This coloring, as described in Adams and Ortega (1982), decouples the equations so that an implementation on either vector or array computers is possible as will become more apparent later in this discussion.

The problem is to determine the displacements, say $u$ and $v$, in the $x$ and $y$ directions respectively at each node in the plate whenever the plate is loaded on one edge and constrained on another. The partial differential equations of plane stress that govern these displacements are well known, see Norrie and DeVries (1978), but do not contribute to the discussion here. The important point to make is that the stiffness matrix $K$ of (1.1) will be symmetric and positive definite and will have dimension $2ab \times 2ab$ where $a$ is the number of rows of nodes and $b$ is the number of columns of unconstrained nodes ($2$ unknowns at each node), and each row of $K$ will contain at most $14$ nonzero elements which correspond to the grid point stencil for linear triangular elements shown in Figure 2.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{grid_point_stencil.png}
\caption{Plate (Triangular Elements)}
\end{figure}

Notice that the values of $a_{max}$ above are the parameters that were given in Table 1, and if no parametrization is desired, these are simply set to one. We also point out that Algorithm 2 can easily be modified to solve problems whose domains are discretized by more complicated finite elements or finite differences as long as a
multicolor ordering is used. For more details see Adams and Ortega (1982). We now turn to the implementation of Algorithm 1 in conjunction with Algorithm 2 on the CYBER 203/205.

3.1 CYBER 203/205 Implementation

On the CYBER 203/205, vectors consist of contiguous storage locations and maximum efficiency of vector operations is achieved for very long vectors. For vectors of length 1000 around 90% efficiency is obtained, but this drops to approximately 50% or less for vectors of length 100 and 10% for vectors of length 10.

To achieve the maximum vector length for our test problem the \( u \) equations at the Red nodes (left to right, bottom to top) including the constrained nodes are numbered first, followed by the corresponding \( v \) equations at the Red nodes, then by the Black \( u \), Black \( v \), Green \( u \), and Green \( v \) equations. The numbering of the constrained equations is necessary for ease of implementation given the CYBER's contiguous storage requirement but also increases the vector length from \( \frac{1}{3}ab \) to \( \frac{1}{3}(ab+1) \). Of course, the actual updating of the storage locations corresponding to these constrained nodes is prohibited by the control vector feature on this machine, see Ortega and Voigt (1977), and for large values of \( a \) and \( b \) little inefficiency is incurred. For a unit square plate, the maximum vector length for our test problem is \( \frac{a}{5} \) and is around 1000 when \( a = 55 \), or equivalently when the width of each triangle is equal to \( 1/54 \).

The contiguous storage requirement coupled with the manner in which the nodes are colored imposes a restriction on the number of nodes that can be in each row of the plate. In particular, the last node in the first row must be Black so that the coloring R/B/G/R/B/G, etc. wraps around from one row to the next.

Now, the calculations of \( K^0u \) and \( Kp_k \) in Algorithm 2 can be done by a straightforward generalization of Madsen, Rodrique, and Karush's [1976] matrix multiplication by diagonals scheme since \( K \) of (3.1) has the structure shown in (3.2) (and will be stored by these diagonals as well):

\[
K = \begin{bmatrix}
R & v & u & v & u & G & v \\
B & v & u & v & u & G & v \\
G & v & u & v & u & G & v \\
\end{bmatrix}
\]

Also, the multiplication of \( B_{c+1}^+e \) and \( B_{c-1}^-e \) in Algorithm 2 can be performed by the same technique. The subtraction in the convergence test \( \|u^{k+1} - u^k\|_w < \epsilon \) vectorizes and the absolute value is performed by the vector absolute value function that is available on the CYBER. The inner products for the calculation of \( a \) and \( b \) are done by a call to an inner product routine which utilizes the machine's vector hardware; however, the additions of the partial sums make this operation considerably slower than the other vector operations required in the algorithm.

Next, we turn to the implementation of Algorithm 1 in conjunction with Algorithm 2 on the Finite Element Machine.

3.2 Finite Element Machine Implementation

The first task for the implementation on this machine is to assign the nodes (and hence equations at the nodes) of the plate to the processors. This is done by assigning each processor, as nearly as possible, an equal number of Red/Black/ and Green unconstrained nodes as illustrated in Figures 3a, 3b, and 3c, where in each figure, the node colorings may repeat beyond the region shown.

![Figure 3a. 18 nodes/processor](image)

![Figure 3b. 9 nodes/processor](image)

![Figure 3c. 3 nodes/processor](image)
In contrast to the CYBER implementation we need not be concerned with numbering the constrained nodes, but instead we should require that each processor receive an equal distribution of each color of the unconstrained nodes.

Since memory is distributed on the Finite Element Machine, each processor stores the portion of \( u \), \( p \), \( T \), \( r \) and \( K \) that corresponds to its collection of nodes. For each equation there is assigned to a processor; 14 storage locations are reserved for the nonzero coefficients of \( K \) that correspond to the grid point stencil in Figure 2. For more information about these data structures see Adams (1982). In addition, storage must be reserved in each processor for the portion of \( p \) that must be received from neighbor processors during the calculation of \( Kp \) each iteration. For example, in Figure 3b, processor 1 must reserve storage for the components of \( p \) that correspond to the 3 border nodes in processor 3 and the 3 border nodes in processor 2, but no components are received from processor 4 since no nodes in processors 1 and 4 share a common triangle. This same storage may be used initially for \( u^0 \) during the calculation of \( Ku^0 \). Similarly, storage must be reserved for the \( r \) components associated with the equations at border nodes in neighbor processors for the multiplications of \( B_j c_j^r \) and \( c_j^r X_j \) in Algorithm 2.

The sending and receiving of the border \( p \) components in each CG iteration in Algorithm 1 and the border \( r \) components during each step of the preconditioner in Algorithm 2 is only (for rectangular regions) between neighbor processors and in particular for our test problem will require six of the machine’s eight nearest neighbor links as shown in Figure 4 for processor \( P_2 \).

![Figure 4. FEM Local Links](image)

Hence, the communication required for the \( m \)-step SSOR preconditioner on this machine is completely local and the amount of data that a given processor must communicate can be seen from Figure 3 to be dependent on its number of neighbors as well as the dimension of the rectangle of nodes assigned to it. To reduce the time required for the I/O, the values of each color to be sent to a given neighbor can be packaged and sent as one record and likewise for the values of a particular color to be received from a given neighbor. If this is done, it becomes advantageous to think of the two equations at the same node as being the same color, because, on this machine, it does not matter that they couple since they will always be assigned to the same processor.

The convergence test in Algorithm 1 is implemented by the signal flag network. Each processor raises its convergence flag whenever its portion of \( p \) values are within the stopping criterion. The processors are then synchronized and tested to see if all flags are raised; if so, the iteration stops — if not, all flags are lowered and the iteration continues.

Lastly, we summarize our remarks about the Finite Element Machine implementation of Algorithm 2 by providing a parallel version in Algorithm 3 that will be executed by processor \( p \). The subscript \( p \) denotes the portion of a vector that is assigned to processor \( p \), the subscript \( n \) denotes the portion of the vector that is received from all of processor \( p \)'s neighbors and the subscript \( t \) denotes the total vector which consists of the components received by, as well as those assigned to, processor \( p \).

\[
\begin{align*}
\mathbf{r}_t &= 0; \quad \mathbf{y}_p = 0 \\
\text{for } s = 1 \to m & \\
& \text{(1) } \mathbf{r}_c = \mathbf{y}_p = 0 \\
& \text{(2) for } c = 1 \to 6 \\
& \text{for } c \equiv 1 \pmod{2} \\
& \text{(1) } \mathbf{x} = -\frac{1}{T} \mathbf{B_j^{c+1}} \mathbf{j}, \mathbf{t} \\
& \text{(2) } \mathbf{D}_{cT} \mathbf{c}_T \mathbf{p} = \mathbf{x} \mathbf{y}_p + \mathbf{a}_m \mathbf{a}_r \\
& \text{(3) } \mathbf{y}_p = \mathbf{x} \\
& \text{(4) if } c \equiv 2 \pmod{2} \\
& \text{send border portion of } \mathbf{r}_c \mathbf{1}_p \text{ to } \mathbf{r}_c \mathbf{1}_p \\
& \text{receive } \mathbf{r}_c \mathbf{1}_n \text{ and } \mathbf{r}_c \mathbf{n}_n \\
& \text{for } c = 1 \to 6 \\
& \text{for } c \equiv 0 \pmod{2} \\
& \text{(1) } \mathbf{x} = -\frac{1}{T} \mathbf{B_j^{c+1}} \mathbf{j}, \mathbf{t} \\
& \text{(2) } \mathbf{D}_{cT} \mathbf{c}_T \mathbf{p} = \mathbf{x} \mathbf{y}_p + \mathbf{a}_m \mathbf{a}_r \\
& \text{(3) } \mathbf{y}_p = \mathbf{x} \\
& \text{(4) if } c \equiv 0 \pmod{2} \\
& \text{send border portion of } \mathbf{r}_c \mathbf{1}_p \text{ to } \mathbf{r}_c \mathbf{1}_p \\
& \text{receive } \mathbf{r}_c \mathbf{1}_n \text{ and } \mathbf{r}_c \mathbf{n}_n \\
\end{align*}
\]

Algorithm 3. FEM \( m \)-step 6-color SSOR

4. Results

The example plane stress problem was run on the CYBER 203 at the NASA Langley Research Center for a unit square plate for varying mesh sizes. Table 2 gives the number of iterations, \( t \), and
time, $T$, in seconds to solve this problem using $m = 0$-$10$. The parametrized preconditioner results are denoted by $P$, the number of rows in the plate by $a$, and the maximum vector length by $v$.

### Table 2. CYBER 203 Iterations and Timings $m$-step SSOR PCG

<table>
<thead>
<tr>
<th>$v$</th>
<th>$a$</th>
<th>$n$</th>
<th>$I$</th>
<th>$T$</th>
<th>$I$</th>
<th>$T$</th>
<th>$I$</th>
<th>$T$</th>
<th>$I$</th>
<th>$T$</th>
<th>$I$</th>
<th>$T$</th>
<th>$I$</th>
<th>$T$</th>
<th>$I$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>8</td>
<td>0</td>
<td>112</td>
<td>.133</td>
<td>157</td>
<td>.213</td>
<td>271</td>
<td>.565</td>
<td>536</td>
<td>3.293</td>
<td>788</td>
<td>11.845</td>
<td>929</td>
<td>22.780</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>11</td>
<td>1</td>
<td>52</td>
<td>.129</td>
<td>66</td>
<td>.184</td>
<td>111</td>
<td>.454</td>
<td>214</td>
<td>2.373</td>
<td>311</td>
<td>7.832</td>
<td>395</td>
<td>17.194</td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>20</td>
<td>2</td>
<td>38</td>
<td>.143</td>
<td>50</td>
<td>.208</td>
<td>79</td>
<td>.478</td>
<td>152</td>
<td>2.428</td>
<td>221</td>
<td>7.773</td>
<td>280</td>
<td>17.380</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1282</td>
<td>80</td>
<td>3</td>
<td>31</td>
<td>.155</td>
<td>39</td>
<td>.216</td>
<td>65</td>
<td>.520</td>
<td>124</td>
<td>2.585</td>
<td>181</td>
<td>8.174</td>
<td>229</td>
<td>18.469</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2134</td>
<td>100</td>
<td>3P</td>
<td>24</td>
<td>.121</td>
<td>30</td>
<td>.167</td>
<td>46</td>
<td>.369</td>
<td>88</td>
<td>1.836</td>
<td>129</td>
<td>5.828</td>
<td>163</td>
<td>13.151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>618</td>
<td>80</td>
<td>5P</td>
<td>19</td>
<td>.143</td>
<td>20</td>
<td>.167</td>
<td>29</td>
<td>.347</td>
<td>56</td>
<td>1.716</td>
<td>82</td>
<td>5.345</td>
<td>104</td>
<td>12.260</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>.159</td>
<td>18</td>
<td>175</td>
<td></td>
<td>25</td>
<td>.348</td>
<td>47</td>
<td>.670</td>
<td>70</td>
<td>5.263</td>
<td>88</td>
<td>12.011</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7P</td>
<td>26</td>
<td>.413</td>
<td>43</td>
<td></td>
<td>1.739</td>
<td>64</td>
<td>5.451</td>
<td>80</td>
<td>12.410</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8P</td>
<td>21</td>
<td>.375</td>
<td>36</td>
<td></td>
<td>1.634</td>
<td>54</td>
<td>5.139</td>
<td>69</td>
<td>11.985</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9P</td>
<td>33</td>
<td>1.660</td>
<td>48</td>
<td>5.056</td>
<td>61</td>
<td>11.731</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10P</td>
<td>31</td>
<td>1.709</td>
<td>44</td>
<td>5.070</td>
<td>55</td>
<td>11.594</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that the inner product routine that was used for these results was developed at Langley and is optimized for the CYBER 203. Several observations can be made from these six test cases.

1. The parametrized preconditioner is better with respect to both the number of iterations and the execution time than the corresponding unparametrized preconditioner.

2. The optimal number of steps of the parametrized preconditioner increased as the vector length increased.

In relation to (2), a interesting question is to determine how many steps would be beneficial for a large problem. The answer to this is quite simple if the number of iterations, $N_m$, could be expressed as a function of $m$, since the execution time of the $m$-step method can be expressed as

$$ T(m) = N_m (A + mB) \quad (4.1) $$

where $A$ is the time for one outer conjugate gradient iteration and $B$ is the time for 1 step of the preconditioner. Now if we assume that $N_{m+1} < N_m$, taking $m+1$ steps is more beneficial than taking $m$ steps whenever

1. $(m+1)N_{m+1} - mN_m < 0$. (This means that the total number of inner loops is less for $m+1$ steps)

or (2) $B/A < \frac{N_m - N_{m+1}}{(m+1)N_{m+1} - mN_m} \quad (4.2)$

The inequalities in (4.2) explain for larger problems when more steps of the preconditioner should be taken. For instance, the values of the left and right side of inequality (2) when $m=9$ are (.81, 1.15), (.68, 3), and (.76, 6) for $a = 41, 62,$ and $80$ respectively. Hence, ten steps are preferable to nine only for $a = 80$.

We now give the Finite Element Machine results. The example plane stress problem with 6 rows and 6 columns of nodes (60 equations) was solved on a 1, 2 and then on a 5-processor Finite Element Machine using the $m$-step SSOR PCG method. For this problem the assignment of unconstrained nodes to the processors is shown in Figure 5.

### Figure 5. FEM Processor Assignments

Observe from Figure 5 that for the two and five processor assignments each processor has an equal number of $R$, $B$, and $G$ nodes as well as an...
equal number of border nodes to be communicated. Therefore, in the absence of communication time and any differences in processor speeds, a speedup of two (five) over the one processor case should be realized.

In regard to (3), if we keep the number of nodes per processor fixed and continue to add processors up to a certain number, say \( n \), the overhead for the preconditioner will still be more than that for the CG method and hence \( \omega \) may become optimal; however, as the number of processors increases beyond \( n \), the value of B/A in (4.2) will continue to decrease until \( m > 4p \) steps of the preconditioner will be optimal. The behavior of the \( m \)-step PCG Algorithm can be modeled as a function of the number of processors, the problem size, and the relative speed of arithmetic to communication times for the machine. For more details, see Adams [1982].

### Table 3. \( m \)-step SSOR PCG Iterations, Timings, Speedups

<table>
<thead>
<tr>
<th>( m )</th>
<th>( p = 1 )</th>
<th>( p = 2 )</th>
<th>Speedup</th>
<th>( p = 5 )</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( I )</td>
<td>( T )</td>
<td>( I )</td>
<td>( T )</td>
<td>( I )</td>
</tr>
<tr>
<td>0</td>
<td>48</td>
<td>63.35</td>
<td>48</td>
<td>33.01</td>
<td>1.92</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>47.90</td>
<td>19</td>
<td>25.85</td>
<td>1.85</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>48.75</td>
<td>13</td>
<td>26.65</td>
<td>1.83</td>
</tr>
<tr>
<td>2P</td>
<td>11</td>
<td>41.95</td>
<td>11</td>
<td>22.95</td>
<td>1.83</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>54.95</td>
<td>11</td>
<td>30.15</td>
<td>1.82</td>
</tr>
<tr>
<td>3P</td>
<td>8</td>
<td>41.25</td>
<td>8</td>
<td>22.75</td>
<td>1.81</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>62.40</td>
<td>10</td>
<td>34.30</td>
<td>1.82</td>
</tr>
<tr>
<td>4P</td>
<td>6</td>
<td>39.80</td>
<td>6</td>
<td>22.00</td>
<td>1.81</td>
</tr>
<tr>
<td>5P</td>
<td>5</td>
<td>40.60</td>
<td>5</td>
<td>22.50</td>
<td>1.80</td>
</tr>
<tr>
<td>6P</td>
<td>5</td>
<td>47.05</td>
<td>5</td>
<td>26.20</td>
<td>1.80</td>
</tr>
</tbody>
</table>

Several observations can be made from Table 3.

1. The effectiveness of the preconditioner as a function of \( m \) was the same for the sequential and two and five processor cases (4p, 5p, 3p, 2p, 1, 2, 3, 4).

2. Taking more than one step of the unparametrized preconditioner was not advantageous.

3. The overhead for the CG(\( m=0 \)) algorithm was less than that for the PCG Algorithm because for two and five processors the communications for the preconditioner rather than for the inner products dominate the overhead.

In regard to (3), if we keep the number of nodes per processor fixed and continue to add processors up to a certain number, say \( n \), the overhead for the preconditioner will still be more than that for the CG method and hence \( \omega \) may become optimal; however, as the number of processors increases beyond \( n \), the value of B/A in (4.2) will continue to decrease until \( m > 4p \) steps of the preconditioner will be optimal. The behavior of the \( m \)-step PCG Algorithm can be modeled as a function of the number of processors, the problem size, and the relative speed of arithmetic to communication times for the machine. For more details, see Adams [1982].

### 5. Summary and Conclusions

The \( m \)-step multicolor SSOR preconditioned conjugate gradient method described herein has been shown to be effective on vector computers and for a small problem was effective on the Finite Element Machine. As more processors and the sum/max hardware circuit become available on this machine, the method will be tested on larger problems. This method does not face the usual difficulty in choosing the optimal relaxation parameter, \( \omega \), for the multicolor SSOR method, since for this ordering and few colors \( \omega = 1 \) is a good choice, see Adams [1983]. A problem still remains in applying the method to irregular regions since the grid must be colored and for array machines must also be distributed to the processors in light of this coloring.

### REFERENCES


Conrad, V., Wallach, Y. [1979]. "Alternating


An M-Step Preconditioned Conjugate Gradient Method for Parallel Computation

Loyce Adams

Institute for Computer Applications in Science and Engineering
Mail Stop 132C, NASA Langley Research Center
Hampton, VA 23665

National Aeronautics and Space Administration
Washington, D.C. 20546

The research reported in this paper was also supported in part under NASA Grant NAG1-46.

Final Report

This paper describes a preconditioned conjugate gradient method that can be effectively implemented on both vector machines and parallel arrays to solve sparse symmetric and positive definite systems of linear equations. The implementation on the CYBER 203/205 and on the Finite Element Machine is discussed and results obtained using the method on these machines are given.

parallel computers
preconditioned conjugate gradient
SSOR (symmetric successive overrelaxation)

Unclassified-Unlimited

64 Numerical Analysis
61 Computer Programming and Software

Unclassified

For sale by the National Technical Information Service, Springfield, Virginia 22161
End of Document