NASA CR-172,150

NASA Contractor Report 172150

NASA-CR-172150
19830020669

ICASE

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
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

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

$$Ku = f \quad (1.1)$$

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, Schrieber [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 $\hat{K}\hat{u} = \hat{f}$ where

$$\hat{K} = Q^T M^{-1} K Q^{-T},\ \hat{u} = Q^T u,\ \hat{f} = Q^{-1} f, \quad (1.2)$$

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

```
(1) Choose $u^0$
(2) $r^0 = f - Ku^0$
(3) $M\hat{r}^0 = r^0$
(4) $p^0 = \hat{r}^0$
(5) For $k = 0,1,\cdots k_{max}$
        (1) $\alpha = \frac{(\hat{r}^k, r^k)}{(p^k, Kp^k)}$
        (2) $u^{k+1} = u^k + \alpha p^k$
        (3) If $\|u^{k+1} - u^k\|_\infty < \varepsilon$ then
            stop, otherwise continue.
        (4) $r^{k+1} = r^k - \alpha K p^k$
        (5) $M\hat{r}^{k+1} = r^{k+1}$
        (6) $\beta = \frac{(\hat{r}^{k+1}, r^{k+1})}{(r^k, r^k)}$
        (7) $p^{k+1} = \hat{r}^{k+1} + \beta p^k$
```

Algorithm 1. PCG Algorithm

We note that the standard conjugate gradient algorithm results by choosing $M = I$.

For vector machines, if $M = I$, 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 α and β. 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, 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 the flag network. However, for a large number of processors, the calculations of α and β 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 $O(\log_2 P)$ 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

The research reported in this paper was supported in part by the National Aeronautics and Space Administration under NASA Grant NAG1-46 while the author was at the University of Virginia, Charlottesville, VA. and in part by the National Aeronautics and Space Administration Contract Nos. NAS1-15810, NAS1-17070 and NAS1-17130 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

N83-28940#

given for a system of equations that results from an example structural engineering problem. Results for this problem on the CYBER 203 and the Finite Element Machine are given in Section 4.

2. M-Step Parallel Preconditioners

2.1 Choosing M

The preconditioned conjugate gradient algorithm of the last section requires a symmetric and positive definite preconditioning matrix M. The question is how to choose M so that the condition number of $\hat{K}$,

$$\kappa(\hat{K}) = \frac{\max_i \lambda_i}{\min_i \lambda_i},$$

is as small as possible.

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

$$M = \frac{\omega}{2-\omega}\left(\tfrac{1}{\omega}D-L\right)D^{-1}\left(\tfrac{1}{\omega}D-U\right) \quad (2.1)$$

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 $M\hat{r} = r$ can be implemented on parallel computers as a forward followed by a backward Multicolor SOR iteration applied to $K\hat{r}=r$ with initial guess $\hat{r}^{(0)}=0$ 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 M that more closely approximates K. If this is done, the resulting preconditioning matrix is

$$M = P\left(I+G+\ldots+G^{m-1}\right)^{-1}. \quad (2.2)$$

Now, M must be symmetric and positive definite to be considered as a preconditioner. The necessary and sufficient conditions for M to satisfy these requirements are given in Adams [1982] and we only note here that if P is 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 corresponded to a Jacobi splitting where P = diag(K).

Even though the preconditioner in (2.2) for the SSOR 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 $\hat{K}$ of (1.2) was proven to decrease as the number of steps of the preconditoner in (2.2) increases; however, the maximum ratio of $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ 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 preconditoner (as results in Section 4 verify). However, by parametrizing this preconditoner, 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 M

Johnson, Micchelli, and Paul [1982] have suggested symmetrically scaling the matrix K to have unit diagonal and then taking m terms of a parametrized Newmann series for $K^{-1} = (I-G)^{-1}$ as the value for M^{-1}. This corresponds to a symmetric preconditioning matrix whose inverse is a polynominal of degree m-1 in G,

$$M_m^{-1} = \alpha_0 I + \alpha_1 G + \alpha_2 G^2 + \ldots + \alpha_{m-1} G^{m-1} \quad (2.3)$$

derived from the Jacobi splitting,

$$K = I - G \quad (2.4)$$

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

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

$$K = P - Q. \quad (2.5)$$

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

$$M_m^{-1} = \left(\alpha_0 I+\alpha_1 G+\alpha_2 G^2+\ldots+\alpha_{m-1}G^{m-1}\right)P^{-1} \quad (2.6)$$

and will be symmetric if P is symmetric. We choose the values of α_i so that the eigenvalues of $M_m^{-1}K$ are positive on the interval $[\lambda_1.\lambda_n]$ that contains the eigenvalues of $P^{-1}K$ and are as 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 α_i that correspond to the SSOR splitting are given in Table 1 for m = 2,3, and 4.

Table 1.
α Values for the m-step SSOR PCG Method

m	α_0	α_1	α_2	α_3
2	1.00	5.00		
3	1.00	-2.00	7.00	
4	1.00	7.00	-24.50	31.50

In the next section we describe how to implement the m-step parametrized SSOR PCG method on the

CYBER 203/205 and on the Finite Element Machine and in Section 4, results on these machines are given.

3. Implementation of the m-step SSOR PCG Method

We first describe the algorithm for solving $M\hat{r} = r$, where M 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.

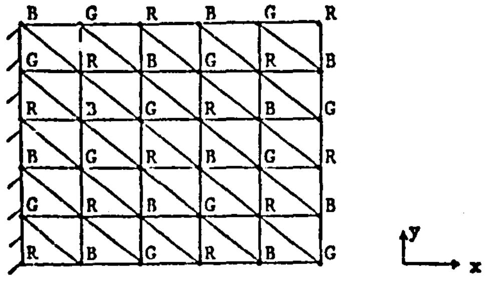


Figure 1. Plate (Triangular Elements)

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.

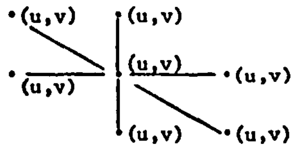


Figure 2. Grid Point Stencil

Observe from Figures 1 and 2 that while there is no coupling between the equations at two nodes of the same color, the equations at a given node do couple. Hence, to completely decouple the system, six colors are necessary; namely, Red(u), Red(v), Black(u), Black(v), Green(u), and Green(v). Now, if the equations at the nodes in Figure 1 are numbered by these six colors from bottom to top, left to right, the system $K\hat{r} = r$ has the form,

$$\begin{bmatrix} D_{11} & B_{12} & B_{13} & B_{14} & B_{15} & B_{16} \\ B_{12}^T & D_{22} & B_{23} & B_{24} & B_{25} & B_{26} \\ B_{13}^T & B_{23}^T & D_{33} & B_{34} & B_{35} & B_{36} \\ B_{14}^T & B_{24}^T & B_{34}^T & D_{44} & B_{45} & B_{46} \\ B_{15}^T & B_{25}^T & B_{35}^T & B_{45}^T & D_{55} & B_{56} \\ B_{16}^T & B_{26}^T & B_{36}^T & B_{46}^T & B_{56}^T & D_{66} \end{bmatrix} \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \\ \hat{r}_3 \\ \hat{r}_4 \\ \hat{r}_5 \\ \hat{r}_6 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{bmatrix} \quad (3.1)$$

where B_{12}, B_{34}, B_{56}, and D_{ii}, $i = 1$ to 6 are diagonal matrices.

The SSOR iteration can be realized by a forward followed by a backward Multicolor SOR iteration, (Adams and Ortega [1982]), but is only as expensive as one Multicolor SOR iteration since a technique of Conrad and Wallach [1979] can be used to save results in an auxiliary vector, y, from the forward pass to be used in the backward pass. The procedure is given below for solving $M\hat{r} = r$ of Algorithm 1. The relaxation parameter ω of the SSOR method causes no problems in the implementation and will be set to one here for simplicity.

```
(1) $\hat{r} = 0$;   $y = 0$
(2) For $s = 1$ to $m$
    (1) For $c = 1$ to 6
        (1) Form $x = -\sum_{j=1}^{c-1} B_{jc}^T \hat{r}_j$
        (2) Solve $D_c\hat{r}_c = x + y_c + \alpha_{m-s} r_c$
        (3) Set $y_c = x$
    (2) For $c = 5$ down to 2
        (1) Form $x = -\sum_{j=c+1}^{6} B_{cj}\hat{r}_j$
        (2) Solve $D_c\hat{r}_c = x + y_c + \alpha_{m-s} r_c$
        (3) Set $y_c = x$
(3) Solve $D_1\hat{r}_1 = -\sum_{j=2}^{6} B_{1j}\hat{r}_j + y_1 + \alpha_o r_1$
```

Algorithm 2. m-step 6-color SSOR

Notice that the values of α_{m-s} 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.

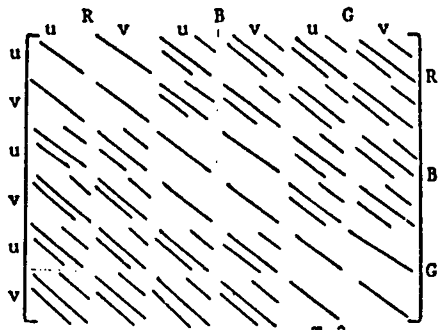
3.1 CYBER 203/205 Implementation

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 1/3ab to $\frac{1}{3}a(b+1)$. 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 $\frac{a^2}{3}$ and is around 1000 when $a \approx 55$, 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 Ku^0 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):

$K =$  (3.2)

Also, the multiplication of $B_{jc}^T\hat{r}_j$ and $B_{cj}\hat{r}_j$ in Algorithm 2 can be performed by the same technique. The subtraction in the convergence test $\|u^{k+1}-u^k\|_\infty < \varepsilon$ 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 α and β 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 conjuction 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.

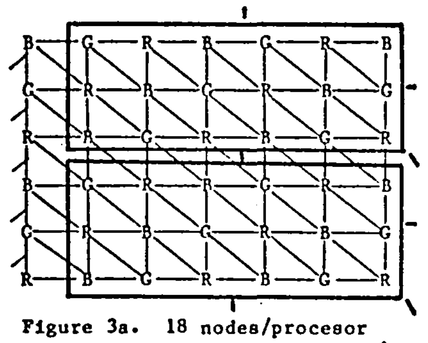


Figure 3a. 18 nodes/procesor

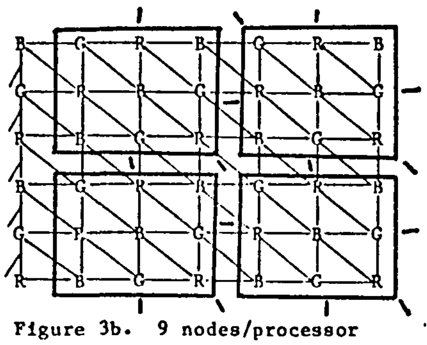


Figure 3b. 9 nodes/processor

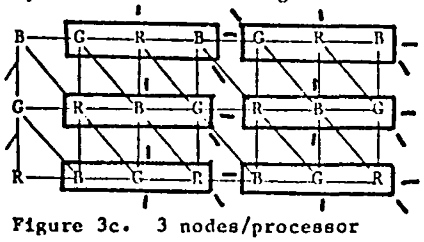


Figure 3c. 3 nodes/processor

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, $\hat{r}$, r and K that corresponds to its collection of nodes. For each equation that 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^o during the calculation of Ku^o. Similarly, storage must be reserved for the $\hat{r}$ components associated with the equations at border nodes in neighbor procesors for the multiplications of $B_{jc}^T\hat{r}_j$ and $B_{cj}\hat{r}_j$ in Algorithm 2.

The sending and receiving of the border p components in each CG iteration in Algorithm 1 and the border $\hat{r}$ 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.

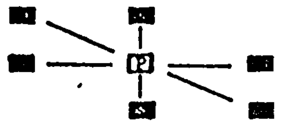


Figure 4. FEM Local Links

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 u 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.

```
(1) $\hat{r}_t = 0;\ y_p = 0$
(2) For $s = 1$ to $m$
    (1) For $c = 1$ to $6$
        (1) $x = -\sum_{j=1}^{c-1} B_{jc}^T \hat{r}_{j,t}$
        (2) $D_{c,p}\hat{r}_{c,p} = x + y_p + \alpha_{m-s} r_p$
        (3) $y_p = x$
        (4) If $c \bmod 2 = 0$ then
            (1) Send border portion of $\hat{r}_{c-1,p}$ and $\hat{r}_{c,p}$
            (2) Receive $\hat{r}_{c-1,n}$ and $\hat{r}_{c,n}$
    (2) For $c = 5$ down to $2$
        (1) $x = -\sum_{j=c+1}^{6} B_{cj} \hat{r}_{j,t}$
        (2) $D_{c,p}\hat{r}_{c,p} = x + y_p + \alpha_{m-s} r_p$
        (3) $y_p = x$
        (4) If $c \bmod 2 \neq 0$ then
            (1) Send border portion of $\hat{r}_{c+1,p}$ and $\hat{r}_{c,p}$
            (2) Receive $\hat{r}_{c+1,n}$ and $\hat{r}_{c,n}$
(3) Solve $D_{1,p}\hat{r}_{1,p} = -\sum_{j=2}^{6} B_{1j}\hat{r}_{1,t} + y_p + \alpha_o r_p$
```

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, I, 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

	v = 22		v = 41		v = 132		v = 561		v = 1282		v = 2134	
	a = 8		a = 11		a = 20		a = 41		a = 62		a = 80	
m	I	T	I	T	I	T	I	T	I	T	I	T
0	112	.133	157	.213	271	.565	536	3.293	788	11.845	929	22.780
1	52	.129	66	.184	111	.454	214	2.373	311	7.832	395	17.194
2	38	.143	50	.208	79	.478	152	2.428	221	7.773	280	17.380
2P	31	.116	40	.167	61	.369	118	1.885	172	6.052	218	13.534
3	31	.155	39	.216	65	.520	124	2.585	181	8.174	229	18.469
3P	24	.121	30	.167	46	.369	88	1.836	129	5.828	163	13.151
4P	22	.138	24	.166	35	.350	67	1.726	99	5.471	124	12.306
5P	19	.143	20	.167	29	.347	56	1.716	82	5.345	104	12.260
6P	18	.159	18	.175	25	.348	47	1.670	70	5.263	88	12.011
7P					26	.413	43	1.739	64	5.451	80	12.410
8P					21	.375	36	1.634	54	5.139	69	11.985
9P							33	1.660	48	5.056	61	11.731
10P							31	1.709	44	5.070	55	11.594

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), an 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

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

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

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,.15), (.68,.5), 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.

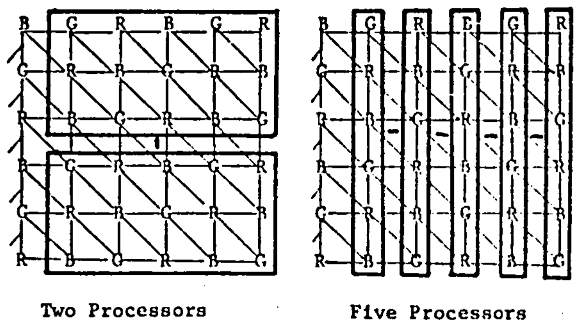


Figure 5. FEM Processor Assignments

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.

The number of iterations and the time in seconds for the above assignments are given in Table 3. The speedups for the two and five processor assignments also are included.

Table 3. FEM Iterations, Timings, Speedups m-step SSOR PCG

	p = 1		p = 2			p = 5		
m	I	T	I	T	Speedup	I	T	Speedup
0	48	63.35	48	33.01	1.92	48	17.70	3.58
1	19	47.90	19	25.85	1.85	19	14.85	3.23
2	13	48.75	13	26.65	1.83	13	15.50	3.15
2P	11	41.95	11	22.95	1.83	11	13.30	3.15
3	11	54.95	11	30.15	1.82	11	17.65	3.11
3P	8	41.25	8	22.75	1.81	8	13.25	3.11
4	10	62.40	10	34.30	1.82	10	20.20	3.09
4P	<u>6</u>	<u>39.80</u>	<u>6</u>	<u>22.00</u>	1.81	<u>6</u>	<u>12.90</u>	3.09
5P	5	40.60	5	22.50	1.80	5	13.25	3.06
6P	5	47.05	5	26.20	1.80			

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_α, the overhead for the preconditioner will still be more than that for the CG method and hence m = 3P or 2P may become optimal; however, as the number of processors increases beyond n_α, 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 modelled 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, ω, for the multicolor SSOR method, since for this ordering and few colors ω = 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

Adams, L., Ortega, J. [1982]. "A Multi-Color SOR Method for Parallel Computation," Proceedings 1982 Conference on Parallel Processing, Bellaire, Michigan.

Adams, L. [1982]. "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," Ph.D. thesis, University of Virginia (Oct. 1982). Also NASA Contractor Report 166027, NASA Langley Research Center.

Adams, L. [1983]. "M-Step Preconditioned Conjugate Gradient Methods." To appear as an ICASE Report.

Chandra, R. [1978]. "Conjugate Gradient Methods for Partial Differential Equations," Ph.D. thesis, Research Report # 129, Department of Computer Science, Yale University.

Concus, P., Golub, G., O'Leary, D. [1976]. "A Generalized conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations," Sparse Matrix Computations, eds. J. Bunch, D. Rose, Academic Press, pp. 309-332.

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

Methods for Sets of Linear Equations," Numerische Mathematik, Vol. 32, pp. 105-108.

Dubois, P., Greenbaum, A., Rodrique, G. [1979]. "Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors," Computing, Vol. 22, pp. 257-268.

Hestenes, M., and Stiefel, E. [1952]. "Methods of Conjugate Gradients for Solving Linear Systems," J. Res. Nat. Bur. Std., pp. 409-436.

Johnson, O., Micchelli, C., Paul, G. [1982]. "Polynominal Preconditioners for Conjugate Gradient Calculations," IBM Research Report 40444, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y.

Jordan, H. [1978]. "A Special Purpose Architecture for Finite Element Analysis," Proc. 1978 Int. Conf. on Par. Proc., pp. 263-266.

Madsen, N., Rodrique, G., Karush, J. [1976]. "Matrix Multiplication by Diagonals on a Vector/Parallel Processor," Information Processing Letters, Vol. 5, No. 2, pp. 41-45.

Norrie, D., DeVries, G. [1978]. An Introduction to Finite Element Analysis, Academic Press, N.Y.

Ortega, J., Voigt, R. [1977]. "Solutions of Partial Differential Equations on Vector Computers," Proc. 1977 Army Num. Anal. Conf., pp. 475-526.

Podsiadlo, D., and Jordan, H. [1981]. "Operating Systems Support for the Finite Element Machine," Computer Science Design Group University of Colorado, Boulder, Colorado.

Reid, J. [1971]. "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations," Proc. Conf. on Large Sparse Sets of Linear Equations, Academic Press, New York.

Schreiber, R. [1981]. "Implementation of the Conjugate Gradient Method on a Vector Computer," Submitted to SIAM Journal on Scientific and Statistical Computation.

1. Report No. NASA CR-172150	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle An M-Step Preconditioned Conjugate Gradient Method for Parallel Computation	5. Report Date June 1983
	6. Performing Organization Code
7. Author(s) Loyce Adams	8. Performing Organization Report No. 83-23
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665	10. Work Unit No.
	11. Contract or Grant No. NAS1-15810, NAS1-17070, and NAS1-17130
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	13. Type of Report and Period Covered Contractor Report
	14. Sponsoring Agency Code

15. Supplementary Notes
The research reported in this paper was also supported in part under NASA Grant NAG1-46.
Langley Technical Monitor: Robert H. Tolson
Final Report

16. 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.

17. Key Words (Suggested by Author(s)) parallel computers preconditioned conjugate gradient SSOR (symmetric successive overrelaxation)	18. Distribution Statement 64 Numerical Analysis 61 Computer Programming and Software Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	9	A02

End of Document