Vol. 7 No. 2 Journal of Computational Mathematics April 1989

LARGE MATRIX COMPUTATIONS ON VECTOR COMPUTERS*

YAU SHU WONG
(*University of Alberta, Edmonton, Canada*)

Abstract

Preconditionings have proved to be a powerful technique for accelerating the rate of convergence of an iterative method. This paper, which is concerned with the conjugate gradient algorithm for large matrix computations, investigates an approximate polynomial preconditioning strategy. The method is particularly attractive for implementation on vector computers.

1. Introduction

In recent years, remarkable progress has been reported on large scale numerical simulations for many practical problems in science and engineering. The rapid advances in the field of computer hardware, in particular, the advent of supercomputer technology, have played a significant role in this development. It is important, however, to note that in order to exploit the full potential of the vector or parallel processors available on supercomputers, numerical algorithms must be developed to take advantage of the specific computer architecture. Many standard numerical algorithms, which have been very successful on the conventional scalar computers, could become inefficient when implemented on the supercomputer.

The purpose of this paper is to study efficient numerical algorithms for large matrix computations on vector processors. It is important to develop very efficient numerical algorithms, because such a problem frequently results from a numerical solution to partial differential equations.

---

* Based on the lecture presented at the China-U.S.A. seminar held at the Xián Jiaotong University, December 1987. This work was supported by the Natural Sciences and Engineering Research Council of Canada Grant U0375.

2. Conjugate Gradient Algorithm

Consider the linear system

$$Au = b, \tag{1}$$

where A is a given symmetric and positive definite matrix. Now, introducing a non-singular matrix M, Equation (1) can be rewritten as

$$AM^{-1}Mu = b. \tag{2}$$

Equation (2) is known as the preconditioned system, and M is the preconditioning matrix.

It has been widely accepted that the method of conjugate gradient (CG) [3] is an efficient iterative technique for large matrix calculations. The CG algorithm for solving Equation (2) can be summarized as follows:

Initialization. Start with an approximation to the solution vector u^0, compute the residual $r^0 = b - Au^0$, and set the direction vector $p^0 = M^{-1}r^0$.

Iteration. For $n = 0, 1, 2, \cdots$, do:

Step 1. $\alpha_n = (r^n, M^{-1}r^n)/(p^n, Ap^n)$.

Step 2. $u^{n+1} = u^n + \alpha_n p^n$.

Stpe 3. $r^{n+1} = r^n - \alpha_n Ap^n$.

Stpe 4. $\beta_n = (r^{n+1}, M^{-1}r^{n+1})/(r^n, M^{-1}r^n)$.

Stpe 5. $p^{n+1} = M^{-1}r^{n+1} + \beta_n p^n$.

The process is continued until $\|r^{n+1}\|$ satisfied a convergence criterion. The inner product (x, y) is defined as $x^T y$ for any vectors x and y.

The CG algorithm presented here can be efficiently implemented on a vector computer. For the $CDC\ CYBER$ 205 computer with 2 pipes and 64-bit arithmetic, the maximum computational rate for a linked triad operation (i.e., vector + constant * vector) is 200 million floating-point operations per second (Mflops). It can be easily seen that Step 2, 3 and 5 are the linked triad operations. The two inner products in Steps 1 and 4 can be computed by the Q8SDOT routine available on the CYBER 205 computer, and the maximum rate is 100 Mflops. The major computational work for each CG iteration consists of the matrix by vector multiplication Ap and the preconditioning step $M^{-1}r$. For large and sparse matrices, the non-zero coefficient elements of A can be stored by diagonals. The computation for Ap can then be efficiently implemented almost entirely using the linked triad operations [6].

The computational work for each CG iteration can be expressed as

$$W = W_b + W_p, \tag{3}$$

where W_b is the work for the basic CG algorithm and

$$W_b = 2 * IP + 3 * LT + 1 * MV. \tag{4}$$

Here IP, LT, and MV denote operations for the inner product, linked triad, and matrix by vector multiplication. W_p is the work resulting from the preconditioning step $M^{-1}r$, and $W_p = 0$ if $M = I$. It is then clear that the basic CG algorithm is suitable for coding on a vector computer, and this leaves only $M^{-1}r$ unvectorized when $M \neq I$.

3. Preconditioning Techniques

The rate of convergence for the CG algorithm can be substantially improved by the application of a good preconditioning matrix M. The important considerations in the selection for M are that M^{-1} should be a good approximation to A^{-1} in some sense, and that the preconditioning step $M^{-1}r$ can be conveniently computed. The first condition ensures that the condition number of the iteration matrix AM^{-1} will be much smaller than that of the original matrix A; hence, it results in a faster convergence rate. The efficiency of the preconditioning depends upon the second condition, in which it is desirable that computing $M^{-1}r$ (i.e., W_p) takes about the same as for W_b.

A popular preconditioning procedure which satisfies these conditions is an incomplete factorization algorithm [7], in which

$$M = LU, \tag{5}$$

where L and U are sparse lower and upper triangular matrices with non-zero elements appearing in the same location as A. The preconditioning step $M^{-1}r$ can then be computed via forward and backward substitutions, and the cost is about the same as one matrix by vector multiplication. This technique has been very successful on a conventional scalar computer, but its implementation is not efficient on a vector processor. This is due to the fact that forward and backward substitutions involve essentially recursive operations, and they are difficult to vectorize. Consequently, only a small computational rate can be obtained for the preconditioning step.

Another approach which remedies this difficulty is to use a truncated Neumann series for approximating a matrix inverse [2]. This is known as a polynomial preconditioning. Suppose the original matrix has been scaled, such that

$$A = I - G, \tag{6}$$

where G is symmetric with zero diagonal entries. If the spectral radius of G is less than one, M^{-1} can then be approximated by a truncated polynomial expansion in G, that is

$$M^{-1} = (I - G)^{-1} \cong \sum_{i=0}^{k} G^i. \tag{7}$$

The preconditioning step $M^{-1}r$ can now be computed via

$$M^{-1}r = (I + G + G^2 + \cdots + G^k)r. \tag{8}$$

To improve the performance, Johnson *et al.* [5] and Saad [8] suggested a parameterized version for (8), in which

$$M^{-1}r = (\gamma_0 I + \gamma_1 G + \gamma_2 G^2 + \cdots + \gamma_k G^k)r, \tag{9}$$

where the parameters γ_i are chosen to minimize the condition number of AM^{-1}.

The computational work due to the preconditioning step $M^{-1}r$ expressed in (8) or (9) is then essentially given by

$$W_p = k * LT + k * MV, \tag{10}$$

if k terms are kept in the polynomial. Since G has the same structure as the original matrix A, the arithmetic operations for Gr needed in W_p is almost the same as that for Ap required in W_b. The computational work for each CG iteration given in Equations (3), (4) and (10) may be dominated by W_p if k is greater than one. The question to be considered now is whether the cost for computing W_p could be reduced, and thus result in an overall improvement for the preconditioned algorithm. The answer is that it is possible to achieve the objective for certain matrix problems.

Suppose a matrix operator $\tilde{A}$ can be found in such a way that $\tilde{A}$ is to some extent related to the original matrix A. However, $\tilde{A}$ consists of far fewer non-zero elements than A. Then an efficient polynomial preconditioning for A can be constructed in which the polynomial expansion is based on $\tilde{A}$ rather than A itself. As a consequence of $\tilde{A}$ has less non-zero coefficients, the computational cost for one matrix by vector multiplication required in the preconditioning step will be less than that needed for Ap in the basic CG algorithm. To distinguish from the traditional preconditioning based on the original matrix operator, this new approach will be referred to as an approximate polynomial preconditioning technique.

4. Numerical Results

In order to illustrate how a matrix operator $\tilde{A}$ can be constructed from a given coefficient matrix A, two problems for large matrix computations will be considered. Numerical experiments have been performed on the $CDC\ CYBER$ 205 computer with 2-pipe and 64-bit arithmetic. For the results reported in this section, the initial vector u^0 is composed of random numbers uniformly distributed on the interval [0,1], and the CG iteration is terminated when $\|r\|_\infty < 10^{-10}$. The preconditioning technique is based on a parametrized polynomial expansion given in (9). The parameters γ_i used for the problems considered in this section can be found in [9] and [10].

Problem 1. Consider the three-dimensional Poisson Equation

$$-\nabla^2 u + gu = f(x, y, z), \quad \text{in } D \tag{11}$$

and $u(x, y, z) = U(x, y, z)$ on ∂D, where $\nabla^2 u = u_{xx} + u_{yy} + u_{zz}$ and $g \geq 0$. A uniform mesh h is used to discretize Equation (11) by a fourth-order compact finite difference scheme [1]. The resulting coefficient matrix has 27 non-zero diagonals, and it has been scaled so that $a_{ii} = 1.0$. Suppose m is the number of grid points in each direction, thus the total number of equations $EQ = m^3$. The arithmetic operation counts for one CG iteration is given by

$$W = (20 + 2k) * m^3 + (k + 1) * (52m^3 - 36m^2 - 12m - 4). \tag{12}$$

The second term represents the major computational work which consists of one matrix by vector multiplication (MV) for the basic algorithm and kMV for the preconditioning step.

Consider a second-order finite difference method is applied to approximate Equation (11), The resulting matrix $\tilde{A}$ will have 7 non-zero diagonals. The arithmetic counts for one MV involving $\tilde{A}$ is then given by $(12m^3 - 4m^2 - 4m - 4)$. Although A and $\tilde{A}$ have different structures (where A has 27 diagonals and $\tilde{A}$ has 7 diagonals), the two matrices are to some extent related since they are both approximating the same partial differential equation. Consequently, a polynomial preconditioning based on $\tilde{A}$ is not only good for the matrix $\tilde{A}$, but it can also be used for A. The arithmetic counts for one CG iteration with this approximate polynomial preconditioning thus becomes

$$\tilde{W} = (20 + 2k) * m^3 + (52m^3 - 36m^2 - 12m - 4) + k * (12m^3 - 4m^2 - 4m - 4). \tag{13}$$

Let EP and AP denote exact and approximate polynomial preconditioning, and ignoring m^2 and other lower-order terms. The computational work for one CG iteration used in conjunction with EP or AP for different values of k is listed in Table 1.

Table 1. Arithmetic operation counts for one CG iteration

k	0	1	2	3
EP	$74m^3$	$126m^3$	$180m^3$	$234m^3$
AP	$74m^3$	$86m^3$	$100m^3$	$114m^3$

The work for the basic CG iteration is given for $k = 0$. Clearly, as the value of k increases, the advantage in using AP instead of the EP method is apparent. Notice that, preconditioning based on AP with $k = 3$ takes less work than EP

with $k = 1$. The number of iterations (NI) and the computing time (CPU) in seconds for the preconditioned CG methods with $k = 3$ for $m = 19, 29, 39$ (i.e., $EQ = 6859, 24389, 59319$) are reported in Table 2.

Table 2. **Values of NI and CPU**

	$EQ = 6859$		$EQ = 24389$		$EQ = 59319$	
Method	NI	CPU	NI	CPU	NI	CPU
EP	22	0.177	30	0.844	41	2.817
AP	27	0.111	38	0.541	51	1.763

As can be seen, even the number of iterations is increased for the AP preconditioning. The overall performance of the CG method with AP is more superior than that based on EP method. In fact, a saving in computing time of more than 35% is obtained when AP is used instead of the EP method for the compact difference equations. The methods have been tested for matrix equations up to $EQ = 262144$, and a similar improvement is obsered [9].

Problem 2. Consider a biharmonic equation

$$\Delta^2 u(x, y) = f(x, y), \text{ in } D \tag{14}$$

with $u(x, y) = U_1(x, y), U_n(x, y) = U_2(x, y)$ on ∂D, where u_n denotes the normal derivative of u, and $\Delta^2 u = u_{xxxx} + 2u_{xxyy} + u_{yyyy}$. Using the standard central finite difference scheme with a uniforni mesh h, the resulting matrix equation A has 13 non-zero diagonals. Even though A is symmetric and positive definite, it does not possess the property of diagonal dominance. It should be noted that, the matrix is very ill conditioned for problem 2, and it has a condition number $0(h^{-4})$ compared to $0(h^{-2})$ for the problem 1.

Suppose A has been scaled so that $a_{ii} = 1.0$. Difficulty may arise when implementing the polynomial preconditioning based on Equations (8) or (9). Since A is not diagonally dominant, the spectral radius of $G, \rho(G)$, could be greater than one. To alleviate this difficulty, the matrix G should be modified [10], such that

$$G = I - \frac{A}{\omega}, \tag{15}$$

whereω is a parameter chosen to ensure that $\rho(G) < 1$. A simple choice for ω is $\omega = \|A\|/2$, where $\|A\|$ is taken to be the maximum row sums of A.

Now, how another coefficient matrix $\tilde{A}$ can be constructed for A? It can be shown that if A is the resulting matrix for a biharmonic equation, then A is related to the difference approximation to a Laplace operator L in such a way

$$A = L^2 + E, \tag{16}$$

where E is a matrix with small rank. It is then natural to consider a polynomial preconditionary based on $\tilde{A} = L$ rather than the original matrix A. Consequently, a substantial reduction in the computational cost for W_p results when k is large. This is due to the fact that A has 13 diagonals, whereas $\tilde{A}$ has 5 diagonals.

The number of CG iterations for $m = 99$ (i.e., $EQ = 9801$) for various numbers of k is shown in Table 3.

Table 3. Number of iterations for $EQ = 9801$

k	0	1	3	5	10	20	30
EP	5492	3024	1712	1203	685	379	261
AP	5492	2335	1002	623	299	142	97

Several interesting phenomena can be observed in these results. First, a large number of iterations is required for the basic CG method (i.e., $k = 0$). This is because A is very ill conditioned. Although the number of iterations for EP and AP methods decreases as k increases, there is no improvement for the computing time when $k \geq 12$. The CG with no preconditioning requires 19.3 seconds for convergence, and it reduces to 13.1 and 2.2 seconds respectively when EP and AP preconditionings are applied. For the biharmonic problems, the computational results reported here indicate that the AP preconditioning is more effective than the EP method. Not only the computing time per iteration is smaller for the AP preconditioning, the rate of convergence is also faster compared to that based on the EP method. It has been proved [4] that the condition number of the iteration matrix AM^{-1} is in fact smaller when the AP preconditioning method is applied. The performance of the basic CG without preconditioning and that with EP and AP preconditionings with $k = 30$ are reported on Table 4, where NI and CPU denote the number of iterations and the computing time in seconds.

Table 4. Values of NI and CPU

	$EQ = 22201$		$EQ = 39601$		$EQ = 62001$	
Method	NI	CPU	NI	CPU	NI	CPU
CG	11673	93.7	19567	280.6	31462	716.1
EP	261	69.4	892	203.5	1470	531.9
AP	182	10.2	297	29.6	442	69.5

These results speak for themselves. It is clear that significant improvement can be achieved when the CG method is applied in conjunction with the AP preconditioning.

It should be pointed out that the iterative methods discussed here could be applied to a more general problem. For instance, the bending of a thin plate in elasticity requires the solution of

$$\Delta^2 u - \delta(T_x u_{xx} + 2T_{xy} u_{xy} + T_y u_{yy}) = f(x, y), \tag{17}$$

where u is the deflexion of the plate, δ is the ratio of the thickness to the rigidity of the plate, T_x, T_{xy} and T_y are the stresses. If δ is small, an approximate polynomial preconditioning described earlier could also be effectively applied for Equation (17).

In this paper, the CG method used in conjunction with a polynomial preconditioning has been demonstrated to be very efficient for large systems of linear equations. The technique could be easily implemented on a vector computer. The idea of an approximate polynomial preconditioning method is introduced here, and it has proved that for certain matrix problems, it is more effective than the standard polynomial preconditioning based on the original coefficient matrix.

References

[1] R. K. Agarwal, A Fourth-order Compact Scheme for Helmholtz Equation in Curvulinear Coordinates, in: R. S. Stepleman, ed., Scientific Computing, North-Holland, Amsterdam, 1983, 147–154.

[2] M. Dubois, A. Greenbaum, G. Rodrigue, Approximate the inverse of a matrix for use in iterative algorithms on vector processors, *Computing*, **22** (1979), 257–268.

[3] M. R. Hestenes, E. Steifel, Method of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Stand.*, **49** (1952), 409–436.

[4] H. Jiang, Ph. D. thesis, University of Alberta, 1988 (in preparation).

[5] O. G. Johnson, C. A. Micchelli, G. Paul, Polynomial preconditioners for conjugate gradient calculations, *SIAM J. Numer. Anal.*, **20** (1983), 362–376.

[6] N. Madsen, G. Rodrigue, J. Karush, Matrix multiplication by diagonals on vector parallel processor, *Inform. Process. Lett.*, **5** (1976), 41–45.

[7] J. A. Meijerink, H. A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Math. Comp.*, **31** (1977), 148–162.

[8] Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, *SIAM J. Sci. Stat. Comp.*, **6** (1985), 865–881.

[9] Y. S. Wong, Solving large elliptic difference equations on CYBER 205, *Parallel Computing*, **6** (1988), 195–207.

[10] Y. S. Wong, H. Jiang, Approximate Polynomial Preconditioning Applied to Biharmonic Equations on Vector Supercomputers, NASA Technical Memorandum 100217, 1987.