The Parallel Computation of Partial Eigensolutions of Large Matrices on a Massively Parallel Processor *

J. Weston[1], M. Szularz[1], M. Clint[2], and K. Murphy[2]

[1] School of Information & Software Engineering, University of Ulster, Coleraine BT52 1SA, Northern Ireland
[2] Department of Computer Science, The Queen's University of Belfast, Belfast BT7 1NN, Northern Ireland

Abstract. The computation of a few eigenvalues and their corresponding eigenvectors of large, usually sparse, real symmetric matrices is often required in the determination of the solution of many of the important problems encountered in scientific and engineering applications. A common characteristic of two distinct classes of iterative methods for the computation of such eigenvalues and eigenvectors is the construction of a sequence of subspaces which contains, in the limit, the desired eigenvectors. In this paper two algorithms, one from each class, for the parallel computation of a few extreme eigenvalues and their associated eigenvectors of large symmetric matrices are discussed. The first algorithm is a simultaneous iteration method in which the subspaces are of a constant dimension; the second is the Lanczos algorithm in which the subspaces increase in dimension. Modified versions of each of the algorithms are proposed and implemented on an MPP Connection Machine CM-200 with 8K processors. A comparative evaluation of the efficiency of the most efficient version of each of the two algorithms for a variety of different types of matrices of maximum order 11,948 is also presented.

1 Simultaneous Iteration Algorithms

Two variations of the simultaneous iteration method of Clint and Jennings [1] for the computation of a few of the largest eigenvalues and associated eigenvectors of a real symmetric matrix are discussed below. The original method, which utilises the Ritz vectors in each iteration subspace, is given in Algorithm 1. A variation of the method in which the Ritz vectors are replaced by approximations, which are much less expensive to compute, is given in Algorithm 2. This algorithm has been shown to be efficient for moderately large matrices ($n \leq 256$) when implemented on an array processor [6].

* This work was supported by the Engineering and Physical Sciences Research Council under grants GR/J41857 and GR/J41864 and was carried out using the facilities of the University of Edinburgh Parallel Computing Centre

1.1 Algorithm 1

Consider a real symmetric matrix A, of order n, whose eigenvalues $\lambda_i, 1 \leq i \leq n$, are such that $\mid \lambda_1 \mid \geq \mid \lambda_2 \mid .. \geq \mid \lambda_n \mid$. Let D_m be the real diagonal matrix, of order m, whose diagonal components $(\lambda_1, \lambda_2, .., \lambda_m)$ yield the required subset of eigenvalues, and let X_m be the orthonormal matrix, of order $n \times m$, whose columns represent the associated eigenvectors $(x_1, x_2, .., x_m)$. Let U_0 be an $n \times m$ orthonormal matrix whose columns represent approximations to these eigenvectors. Generate a sequence of eigenvector approximations U_k as follows :

$$V_k = AU_k \;\; ; \;\; B_k = U_k^t V_k \;\; ; \;\; U_{k+1} = \mathbf{ortho}(V_k \mathbf{transform}(B_k))$$

where :

(i) the function **transform** returns a matrix T_k whose columns are the eigenvectors of B_k. Note that any appropriate algorithm may be used in the computation of the eigensolution of B_k.

(ii) the function **ortho** generates from its matrix argument a matrix whose normalised columns are mutually orthogonal. It is implemented using a modified Gram-Schmidt algorithm [3] in which the columns are processed, not in strictly left-to-right order, but are processed according to decreasing order of magnitude in the eigenvalue approximations to which they correspond (namely, $\mathrm{diag}(\theta_1^{(k)}, \ldots, \theta_m^{(k)}) = T_k^t B_k T_k$).

Then $U_k \rightarrow X_m$ and $B_k \rightarrow D_m$.
Observe that the convergence of the algorithm depends on the ratio $\mid \lambda_{m+1}/\lambda_m \mid$. Thus, the closer this ratio is to unity, the poorer the convergence rate will be. Moreover,

$$\mid \theta_i^{(k)} - \lambda_i \mid = \mathcal{O}(\mid \frac{\lambda_{m+1}}{\lambda_i} \mid^k) \qquad i = 1 : m$$

Hence, if p eigenvalues are required it can be advantageous, particularly on an MPP machine, to choose the dimension of the iteration subspace m to be greater than p. Thus, in this paper, versions of Algorithm 1 in which m is greater than p are critically examined.

1.2 Algorithm 2

Versions of the above algorithms have been developed in which the function **transform** is replaced by another function **eigenapprx** which generates from its symmetric argument B_k a non-orthogonal matrix whose columns are approximations to the eigenvectors of B_k. This approximation, T_k, with components denoted by $t(i,j)$, is defined as follows

$$t(i,j) = 1, i = j \;\; ; \;\; t(i,j) = e(i,j), i > j \;\; ; \;\; t(j,i) = -t(i,j), i < j$$

where

$$e(i,j) = 2b(i,j)/(b(j,j) - b(i,i) \pm \sqrt{(b(j,j) - b(i,i))^2 + 4b(i,j)^2})$$

and $b(i,j)$ is the i,j-th component of the matrix B_k.
Algorithm 2 is a version of Algorithm 1 in which the function **transform** is replaced by the function **eigenapprx**.
The question now arises as to whether the performance of Algorithm 1 in an MPP environment for large matrices is better than that of Algorithm 2.

1.3 Convergence Criteria for Algorithms 1 and 2

The accuracy of an eigensolver is often measured by the residual error and the orthogonality error where these errors are given by :

$$\text{residual error} = \mathcal{R} = \max_i \frac{\| Ax_i - \lambda_i x_i \|_2}{\lambda_i}$$

and

$$\text{orthogonality error} = \mathcal{O} = \max_i \| (X_p^t X_p - I)e_i \|_2$$

where $i = 1 : p$ and $p \leq m$. Since $V_k = AU_k$ has to be computed in each iteration it follows that $\mathcal{R}$ can be computed with little extra cost. Consequently,

$$\mathcal{R} \leq tol$$

is used as the stopping criterion for the simultaneous iteration algorithms, where tol is related to machine accuracy.

2 Algorithm 3 : The Lanczos Algorithm

Let A be a real symmetric matrix of order n and let q_1 be an arbitrary vector of length n which satisfies $\| q_1 \|_2 = 1$. Then the 'standard' Lanczos algorithm generates a sequence of tridiagonal Lanczos matrices $T_j \in \Re^{j \times j}$ and Lanczos vectors $q_j \in \Re^n$ with the properties that $T_{j-1} \in \Re^{j-1 \times j-1}$ is a principal submatrix of T_j, $T_j = Q_j^T A Q_j$ where $Q_j = [q_1, \ldots, q_j]$ is orthonormal, and for $j \ll n$, the extreme eigenvalues of A are well approximated by the eigenvalues of the Lanczos matrices T_j. In the j-th step the accuracy of these approximations is determined by computing at least a partial eigensolution of the current Lanczos matrix, T_j and monitoring the last elements of each of its eigenvectors [4]. Although the overhead associated with this convergence monitoring process is assumed to be inexpensive in comparison with the other operations of the Lanczos algorithm, it can be quite significant in terms of the absolute time when j is large, say of order 10^3. However, a new and efficient convergence monitoring process which substantially reduces this overhead has been proposed by Szularz et al [7]. Algorithm 3 is a version of the Lanczos algorithm which incorporates this new convergence monitoring process, a brief outline of which appears elsewhere in these proceedings [5].
It has been shown already that Algorithm 3 is more efficient than the 'standard' Lanczos algorithm when implemented both in a shared-memory environment [7], and in an MPP environment [8].

3 Numerical Experience

The performances on a Connection Machine CM-200 with 8K processors of Algorithms 1 and 2 for the partial eigensolution of a variety of positive definite matrices taken from the Harwell-Boeing collection [2] have been compared and analysed. Further, partial eigensolutions have also been obtained using versions of these algorithms in which the number of eigenvector approximations used, m, exceeds the number of maximal eigenvalues required, p. In addition the performances of the implementations of the best versions of Algorithms 1 and 2 were compared with the performance of Algorithm 3.
All of the algorithms have been implemented in CM Fortran 90 using single precision arithmetic.
In the case of Algorithm 3, the Lanczos algorithm, the prescribed tolerance, tol_1, was assigned a value $tol_1 = \mathbf{c}\| A \|_2\mathbf{u}$, where $\mathbf{c}$ is a user-defined constant, $\| A \|_2$ is approximated by λ_1, and $\mathbf{u}$ is the machine unit round-off. In the case of the CM-200 $\mathbf{u} = 0.11920929 \times 10^{-6}$. Recall that in the case of the simultaneous iteration algorithms the convergence criterion required that $\mathcal{R}$, the residual error, be less than a prescribed tolerance, tol, which is related to machine accuracy. Thus, in order to compare the simultaneous iteration algorithms with the Lanczos algorithm, tol is assigned the value of $\mathbf{c}_1\mathcal{R}$, where $\mathbf{c}_1$ is a constant and $\mathcal{R}$ is explicitly computed using the partial eigensolutions obtained using Algorithm 3. In the case of Algorithm 3 $k = 4096$ and $q_1 = [1/\sqrt{n}, \ldots, 1/\sqrt{n}]$. The initial approximations U_0 in the simultaneous iteration algorithms are *either* Q_m *or* $\breve{U}$, where Q_m is the matrix of the first m Lanczos vectors and $\breve{U}$ is the matrix $[q_1, e_2, \ldots e_m]$, with orthonormalized columns.
In the examples which follow n denotes the order of the target matrix A, nz denotes the number of upper (lower) triangular entries of A, and $\kappa(A)$ denotes the condition number of A. The examples are taken from the Harwell-Boeing collection of sparse matrices and are identified by means of character strings.

3.1 Implementation

In the case of Algorithms 1 and 2 all matrices are stored as CM (distributed) arrays thereby enabling advantage to be taken of the highly optimized CMSSL routines. In particular, the use of the CMSSL Jacobi routine to compute the eigensolutions of the distributed matrices B_k in Algorithm 1 is most efficient since m is usually reasonably large and the matrices B_k tend to diagonal form. However, in those cases where m is very small a better option would be to compute the eigensolutions of these matrices on the front-end machine and to use the appropriate routines from the LAPACK library.
Observe that the function **eigenapprx** in Algorithm 2 has been designed for efficient implementation on MPP machines [6].
In the case of Algorithm 3 only the elements of the matrix T_j are stored in the front-end machine ; all other arrays are involved in highly parallel computations and, consequently, are stored as CM arrays.

3.2 Example 1

Matrix NOS3 ($n = 960$, $nz = 8402$, and $\kappa(A) = 3.5 \times 10^3$) is taken from the field of structural engineering and arises in the context of a finite element approximation to a biharmonic operator on a rectangular plate with one side fixed and the others free. Algorithm 3 was used to compute a partial eigensolution of NOS3 and yielded the following results when the parameters **c** and p were assigned the values of 1 and 16, respectively :

(i) $\lambda_1 = 0.68990387 \times 10^3$; computed in 48 steps/iterations.
(ii) $\lambda_{16} = 0.62797619 \times 10^3$; computed in 150 steps/iterations.
(iii) eigenvalues $\lambda_1, \ldots, \lambda_{16}$ were computed in 14.496 sec.
(iv) $\mathcal{R} = 0.152 \times 10^{-6}$; $\mathcal{O} = 0.369 \times 10^{-5}$.

Table 1 presents the results obtained for various values of m when Algorithms 1 and 2 were used to compute the 16 largest eigenvalues of matrix NOS3. The entries in the second and fourth rows refer to Algorithm 1 and the boldface entries in the third and fifth rows refer to Algorithm 2. For each each algorithm U_0 and *tol* were assigned the values Q_m and $\mathcal{R}$, respectively, where $\mathcal{R}$ is given in (iv) above.
Additionally, for this matrix, $\max_m \mathcal{O} = 0.158 \times 10^{-5}$.

m	16	24	32	40	48	56
iterations	1290	167	102	66	54	47
	747	**164**	**92**	**66**	**57**	**104**
time (seconds)	413	81	63	59	59	62
	198	**69**	**50**	**54**	**58**	**130**

Table 1. (Simultaneous Iteration - Example 1)

3.3 Example 2

Matrix NOS7 ($n = 729, nz = 2673$ and $\kappa = 1.8 \times 10^9$) arises in the finite difference approximation to a diffusion equation with varying diffusivity in a 3D unit cube with Dirichlet boundary conditions. Algorithm 3 was used to compute the 16 largest eigenvalues of this matrix.
In the case when $\mathbf{c} = 1.0$ it was observed that four separate clusters of eigenvalues - (λ_5, λ_6), (λ_8, λ_9), $(\lambda_{12}, \lambda_{13})$ and $(\lambda_{14}, \lambda_{15}, \lambda_{16})$ - were identified as four sets of multiple eigenvalues. Thus, for example, λ_{12} and λ_{13} were both located in the interval $(\theta_{923}, \theta_{924})$, where $(\theta_{924} - \theta_{923}) = tol_1$. When **c** was assigned the value 0.1 the following results were obtained :

(i) $\lambda_1 = 0.98640301 \times 10^7$; computed in 19 steps/iterations.

(ii) $\lambda_{16} = 0.52361851 \times 10^7$; computed in 114 steps/iterations.
(iii) $\lambda_1, \ldots, \lambda_{16}$ were computed in 10.416 sec.
(iv) $\mathcal{R} = 0.262 \times 10^{-6}$; $\mathcal{O} = 0.442 \times 10^{-6}$.

Further, the separation of the 16 eigenvalues was at least 0.11758840.
When Algorithm 1 is used to compute a partial eigensolution of matrix NOS7 with the parameters m, p, $\mathbf{c}_1$ and U_0 assigned the values 16,4, 10^3 and Q_{16}, respectively, the four largest eigenvalues are identified as $\lambda_1, \lambda_2, \lambda_5$ and λ_9. Further, within the same limits of precision, it is possible to compute the nine largest eigenvalues correctly if U_0 is assigned the value $\breve{U}$. Table 2 presents the results obtained for various values of m and tol when Algorithm 1 was used to compute the 16 largest eigenvalues of this matrix with U_0 assigned the value $\breve{U}$. The principal entries in rows 2 - 4 represent the number of steps required for convergence and the entries in brackets are the times taken in seconds. The entry '-' indicates that the algorithm failed to converge in fewer iterations than its predesessor in the same row. The value of $\mathcal{R}$ is given in (iv) above.
For the same problem Algorithm 2 failed to converge in fewer than 500 iterations for all choices of m in Table 2. Further, very slow convergence has been observed when Algorithm 2 is used to compute as few as four eigenvalues of matrix NOS7 with much relaxed tolerances.

m	16	24	32	40	48	56
$tol = 10^2\mathcal{R}$	379(104)	17(8)	10(6)	9(8)	9(10)	-
$tol = 10\mathcal{R}$	499(138)	23(9)	12(8)	11(10)	10(12)	-
$tol = \mathcal{R}$	610(165)	-	-	-	-	-

Table 2. (Simultaneous Iteration (Algorithm 1) - Example 2)

3.4 Example 3

Matrix BCSSTK18 ($n = 11,948$, $nz = 80,519$) models a static problem arising in an application of the GT-STRUDL structural engineering code related to the construction of the R.E. Ginna Nuclear Power Station. Algorithm 3 was used to compute a partial eigensolution of BCSSTK18 and yielded the following results when parameters $\mathbf{c}$ and p were assigned the values of 1 and 8, respectively :

(i) $\lambda_1 = 0.42951982 \times 10^{11}$; computed in 16 steps/iterations.
(ii) $\lambda_8 = 0.25322804 \times 10^{11}$; computed in 63 steps/iterations.
(iii) eigenvalues $\lambda_1, \ldots, \lambda_8$ were computed in 8.398 sec.
(iv) $\mathcal{R} = 0.330 \times 10^{-6}$; $\mathcal{O} = 0.262 \times 10^{-6}$.

In this example all of the required eigenvalues are well separated. Thus, the simultaneous iteration algorithms should, in theory, benefit from the use of $U_0 = Q_8$ as an initial approximation. The results obtained for various choices for m and tol when Algorithm 1 was used to compute the 8 largest eigenvalues of matrix BCSSTK18 with U_0 assigned the value Q_8 are shown in Table 3. The principal entries in rows 2 and 4 represent the number of steps required for convergence and the entries in brackets in rows 3 and 5 are the corresponding execution times in seconds. The entry '-' indicates that the algorithm failed to converge in fewer than 2,000 iterations. The value of $\mathcal{R}$ is given in (iv) above. Additionally, for this matrix, $\max_m \mathcal{O} = 0.238 \times 10^{-6}$.

m	8	16	24	32	40	48	56
$tol = 10^3\mathcal{R}$	130	39	28	21	11	3	3
	(220)	(97)	(89)	(86)	(60)	(19)	(22)
$tol = \mathcal{R}$	-	72	57	45	27	13	13
		(183)	(183)	(184)	(150)	(85)	(97)

Table 3. (Simultaneous Iteration (Algorithm 1) - Example 3)

For the same problem Algorithm 2 failed to converge for all choices of m in Table 3 except for $m = 8$. In this instance the value of tol was $10^3\mathcal{R}$, the number of iterations required for convergence was 100, and the time was 201 seconds.

4 Conclusions

In the case of the simultaneous iteration algorithms (Algorithms 1 and 2) the results presented enable the following observations to be made :

(i) In all cases Algorithm 1 computed the required subset of eigenvalues to machine accuracy. However, it was not possible to predict in advance the smallest value of m which would guarantee convergence.

(ii) Whenever the accuracy requirements were less stringent than in (i) Algorithm 1 converged for all selected values of m. Further, in these circumstances the trade-off between the speed of convergence and amount of computation in this MPP environment is justified if m is moderately larger than p. However, the algorithm may become unstable if m is large.

(iii) The use of $\breve{U}$ (non Krylov vectors) as an initial approximation to the eigenvectors in Algorithm 1 was less efficient than the use of Q_m (Krylov vectors) except when the target matrix contained multiple or closely clustered eigenvalues.

(iv) In general Algorithm 1 can use the hardware to advantage providing that a suitable choice of m can be made.

(v) Algorithm 2 is not stable as is demonstrated by Examples 2 and 3 even though, in certain cases, it performs substantially better than Algorithm 1. Since it is not known at present what causes the loss of stability (Example 2 suggests it is due to problem ill-conditioning, closely clustered/multiple eigenvalues or a combination of both) Algorithm 2 must be used with caution.

4.1 Algorithm 1 versus Algorithm 3

On the basis of the problems which arise in 'real' world applications and which have been used to test the methods discussed, the Lanczos algorithm (Algorithm 3) performs significantly better than Algorithm 1. In addition, what was unexpected, the orthogonality among the computed eigenvectors is as good as that obtained by the use of Algorithm 1. In general, the numerical properties of Algorithm 1 inhibit it from gaining much advantage from the hardware by having $m > p$. For the Lanczos algorithm the MPP machine is preferable to a shared memory machine, mainly because of the efficiency with which the matrix-vector product can be performed.

References

1. Clint, M., and Jennings A., The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration, Comput. J., (1970) 76-80.
2. Duff, I.S., Grimes, R.G., and Lewis, J.G., User's Guide for the Harwell-Boeing Sparse Matrix Collection (release I), available online ftp orion.cerfacs.fr, (1992).
3. Golub, G, and Van Loan C.F., Matrix Computations, John Hopkins University Press, London, (1989).
4. Kim, S.K, and A.T. Chronopoulos, A class of Lanczos-like algorithms implemented on parallel computers, Parallel Computing, 17 (1991) 763-778.
5. Murphy, K., Clint, M., Szularz, M., and Weston, J.S., The Computation of Partial Eigensolutions on a Distributed Memory Machine Using a Modified Lanczos Method, Proceedings Euro-Par'96, (1996).
6. Stuart, E.J., and Weston J.S., An Algorithm for the Parallel Computation of Subsets of Eigenvalues and Associated Eigenvectors of Large Symmetric Matrices using an Array Processor, in Proceedings Euromicro Workshop on Parallel and Distributed Processing, 27-29 January, 1993, Milligan, P., and A. Nunez (Eds.), IEEE Computer Society Press (1992), 211-217.
7. Szularz, M., Weston, J., Murphy, K., and Clint M., Monitoring the Convergence of the Lanczos Algorithm in Parallel Computing Environments, Parallel Algorithms and Applications, Vol.6, (1995), 287-302.
8. Szularz M., Weston J., Murphy K. and Clint M., An Improved Lanczos Algorithm, Abstracts : ParCo 95, Gent, Belgium, 19-22 September 1995, p.113.