Rothe, K.

A parallel algorithm for the generalized symmetric eigenvalue problem on a transputer network

Conventional routines from standard librarys (e.g. EISPACK) for solving the generalized symmetric eigenvalue problem $Ax = \lambda Bx$ cannot work efficiently on parallel computers with distributed memory because of their sequential conception. The algorithm presented here parallizes the given problem very efficiently on a transputer network.

1. The sequential standard algorithm

We want to compute dense symmetric $n \times n$ matrix eigenvalue problems with positive definite B that arise in the discretisation and subsequent condensation of a boundary eigenvalue problem (cf. 4).

	generalized problem	$Ax = \lambda Bx$	
1.	Cholesky decomposition	$LL^T := B$	REDUC
2.	reduction	$LC := A$	
3.		$\tilde{A}L^T := C$	
	standard problem	$\tilde{A}y = \lambda y$	
4.	Householder reduction	$T := H^{(n-2)} \cdots H^{(1)} \tilde{A} H^{(1)} \cdots H^{(n-2)}$	TRED1
	tridiagonal problem	$Tz = \lambda z$	
5.	eigenvalue computation		TRIDIB
6.	eigenvector computation		TINVIT
7.	backtransformation	$y = H^{(n-2)} \cdots H^{(1)} z$	TRBAK1
8.		$y = L^T x$	REBAK

Table 1. Standard algorithm for solving $Ax = \lambda Bx$

The standard algorithm for computing a small number of eigenvalues in the lower part of the spectrum with corresponding eigenvectors is given in Table 1 with matching EISPACK routines (cf. 2).

2. The process topology

For the parallel version of the standard algorithm we take a ring of processes since we want to use already existing and efficiently working algorithms for the Cholesky decomposition (cf. 3) and for the Householder reduction (cf. 1).

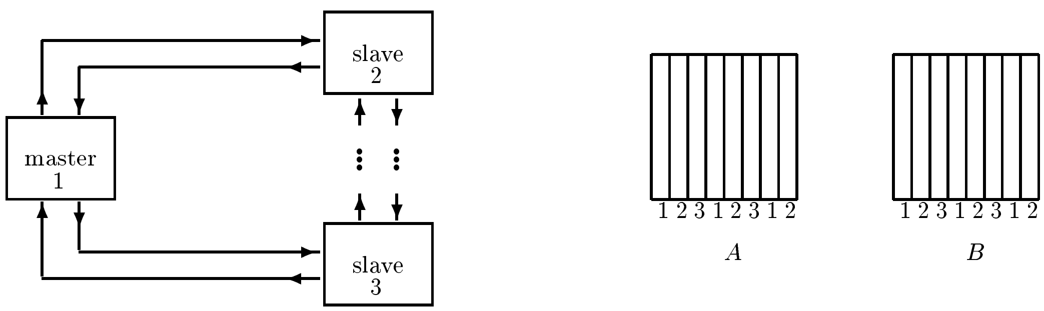

Figure 1. Ring of three (or more) processes with wrapped mapping of the columns of A and B

Each process can be identified with a corresponding driver routine named 'slave' respectively 'master' which is responsible for the initialisation. By the cdl-compiler of the distributed memory operating system 'helios' of the transputer network these drivers are connected to the bidirectionally communicating ring with distributed memory as given in Figure 1. The mapping of the ring structure to the network of T800 transputers is assumed to be optimal and will not be discussed here.

3. The parallel version of the standard algorithm

We now parallize the eight computational steps of the standard algorithm by eight corresponding subroutines which are called simultaneously step by step by the driver routines. The main feature of an efficient parallel algorithm is the data handling. In our case we start with the wrapped mapping of the columns of A and B which is given in Table 1. The algorithm then continues with the wrapped mapping of the columns which is shown in Table 2. It should be pointed out however, that data restructuring between steps must be avoided. Therefore the data output of one step has to be the data input for the next step, thus making the calculation of the full matrix C (instead of the lower triangle) in the subroutine FORWARD necessary. The consequence is a little loss in the speedup relative to EISPACK which is documented in Table 4.

	name of subroutine	data input (columns)	data output (columns)
1.	CHOLESKY	B wrapped mapping	L wrapped mapping (lower triangle of B)
2.	FORWARD	A, L wrapped mapping	C wrapped mapping (on A)
3.	ALTEQC	C, L wrapped mapping	$\tilde{A}$ wrapped mapping (lower triangle of A)
4.	HOUSEHOLDER	$\tilde{A}$ wrapped mapping	H, T wrapped mapping ($H^{(i)}$ lower triangle of A)
5.	EIGENVALUES	T wrapped mapping	T each process
6.	EIGENVECTORS	T each process	block of eigenvectors on each process
7.	TRANSBAK1	$H^{(i)}$ wrapped mapping	block of eigenvectors on each process
8.	TRANSBAK2	L wrapped mapping	block of eigenvectors on each process

Table 2. Data distribution of the parallel version of the standard algorithm

For a more detailed description of the algorithm we refer to a technical report that will appear. Table 3 shows the very high efficiency of the first five steps of the algorithm for computing the twentyfive smallest eigenvalues with $p = 1, \cdots, 4$ processes.

$n\,/\,p$	1	2	3	4
100	100.0	88.7	74.0	60.8
200	100.0	95.7	89.3	81.5
300	100.0	97.6	93.6	88.8
400	100.0	98.3	95.7	92.2

Table 3. Efficiency of CHOLESKY - EIGENVALUES in per cent

$n\,/\,p$	1	2	3	4
100	0.658	1.168	1.461	1.601
200	0.646	1.237	1.731	2.078
300	0.643	1.255	1.805	2.283
400	0.642	1.262	1.844	2.369

Table 4. Speedup CHOLESKY - EIGENVALUES compared with REDUC - TRIDIB

Table 4 compares the presented algorithm with the corresponding EISPACK sequence and shows that the computational time for $n = 400$ and $p = 4$ is more than halved.

If the eigenvectors are requested also one has to activate the last three steps of the algorithm. The efficiency and the speedup will then be nearly the same as before since the first four steps dominate the computational effort.

4. References

1 Dongarra, J.J.; van de Geijn, R.A.: Reduction to condensed form for the eigenvalue problem on distributed memory computers. Parallel Computing **18** (1992), 973–982.

2 Garbow, B.S.; Boyle, J.M.; Dongarra, J.J.; Moler, C.B.: Matrix eigensystem routines – EISPACK Guide Extension. Lecture notes in computer science **51**, Springer,New York, 1977.

3 Golub, G.H.; van Loan, C.F.: Matrix Computations. The John Hopkins University Press, Baltimore and London, 1989.

4 Rothe, K.; Voss, H.: Improving dynamic substructuring on distributed memory MIMD-Computers. In: Brebbia, C.A.; Power, H. (eds.): Applications of Supercomputers in Engineering III, Computational Mechanics Publications, Southhampton, Boston and Elsevier Science Publishers, London, New York, 1993, 339–352

Adress: ROTHE, K., University of Hamburg, Institute for Applied Mathematics, Bundesstrasse 55, D-20146 Hamburg