A Parallel Lanczos Algorithm for Eigensystem Calculation

Hans-Peter Kersken / Uwe Küster

Eigenvalue problems arise in many fields of physics and engineering science for example in structural engineering from prediction of dynamic stability of structures or vibrations of a fluid in a closed cavity. Their solution consumes a large amount of memory and CPU time if more than very few (1-5) vibrational modes are desired. Both make the problem a natural candidate for parallel processing. Here we shall present some aspects of the solution of the generalized eigenvalue problem on parallel architectures with distributed memory.

The research was carried out as a part of the EC founded project ACTIVATE. This project combines various activities to increase the performance vibro-acoustic software. It brings together end users form space, aviation, and automotive industry with software developers and numerical analysts.

Introduction

The algorithm we shall describe in the next sections is based on the Lanczos algorithm for solving large sparse eigenvalue problems. It became popular during the past two decades because of its superior convergence properties compared to more traditional methods like inverse vector iteration. Some experiments with an new algorithm described in [Bra97] showed that it is competitive with the Lanczos algorithm only if very few eigenpairs are needed. Therefore we decided to base our implementation on the Lanczos algorithm. However, when implementing the Lanczos algorithm one has to pay attention to some subtle algorithmic details. A state of the art algorithm is described in [Gri94]. On parallel architectures further problems arise concerning the robustness of the algorithm. We implemented the algorithm almost entirely by using numerical libraries. All the numerical work is performed using either PETSc [Bal97], LAPACK [And95], or BLAS [Don88].

The Lanczos Algorithm

Here we shall introduce the principals of the Lanczos algorithm for the simple case of the standard eigenvalue problem $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$. It works by constructing an orthonormal basis of a Krylov-subspace and calculates the Ritz approximation of the eigenpairs in this subspace. A Krylov-subspace is spanned by the vectors obtain from repetitive application of the operator to an initial vector, i. e. by $\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \dots\}$. By the properties of the Krylov-subspace the problem reduces to find the eigenpairs of a relatively small tridiagonal matrix. The algorithm to calculate the eigenpairs of an operator $\mathbf{A}$ is given in the box 1. The $\mathbf{q}_j$ are called the Lanczos vectors, $(\Theta_i, \mathbf{s}_i)$ are the eigenpairs of the tridiagonal matrix, and $(\Theta_i, \mathbf{y}_i)$ are the Ritz pairs, i. e. the approximation to eigenpairs of $\mathbf{A}$ from the subspace. The tridiagonal matrix consists of the α_j on the diagonal and the β_j on the first sub- and superdiagonal. At each step a new Ritz pair is added and some of them start to converge to eigenpairs of $\mathbf{A}$. The iteration is stop if all eigenvalues in a given interval are found. Due to the convergence properties of the Lanczos algorithm the eigenvalues at both ends of the spectrum converge first. Additionally the convergence rate depends on the relative separation $\left|\lambda_i - \lambda_{i+1}\right| / \left|\lambda_n - \lambda_1\right|$ of the eigenvalues. Two problems arise when this simple approach is used. In finite precision arithmetics the Lanczos vectors eventually loose the property of been orthogonal resulting in a recalculation of already computed eigenpairs. The second is connected to the distribution of eigenvalues in vibrational problems in structural analysis. Here the smallest eigenvalues are of interest but have a small relative separation compared to the largest especially because the spread of the spectrum is usually quite large. The relative separation of the smallest eigenvalues can be in of the order 10-10. Therefore many of the largest eigenvalues will be

calculated before the Ritz pairs will start to converge to small eigenpairs as well. In the following two sections we shall discuss remedies for these problems.

chose $\mathbf{r}_0$
$\beta_0 = \|\mathbf{r}_0\|$
$j = 0$
repeat
  $j = j + 1$
  $\mathbf{q}_j = \mathbf{r}_{j-1}\beta_{j-1}$
  $\mathbf{u}_j = \mathbf{A}\mathbf{q}_j$
  $\mathbf{r}_j = \mathbf{u}_j - \mathbf{q}_{j-1}\beta_{j-1}$
  $\alpha_j = \mathbf{q}_j \ \mathbf{r}_j$
  $\mathbf{r}_j = \mathbf{r}_j - \mathbf{q}_j\alpha_j$
  $\beta_j = \|\mathbf{r}_j\|$
  compute $\Theta_i, \mathbf{s}_i, \mathbf{y}_i, i = 1, \ldots, j$

Box 1: Simple Lanczos

Orthogonalization

Since Paige [Pai72] showed that the loss of orthogonality comes in finite precision arithmetics always hand in hand with the convergence of an eigenpair some means for maintaining orthogonality has been conceived. The most simple and most expensive approach is the complete orthogonalization. Every newly computed Lanczos vector is orthogonalized against all previously computed ones. This is an increasingly expensive process ($O(n^2)$) in the course of the Lanczos algorithm and retains orthogonality up to machine precision ε. Parlett and Scott [Par79] showed that it is sufficient to maintain what is termed semi-orthogonality. This means that the dot product of two Lanczos vectors can be as large as $\sqrt{\varepsilon}$ without affecting the accuracy of the final results. They proposed the selective orthogonalization scheme. Here the relation between converging eigenpairs and loss of orthogonality is used to check the orthogonality and orthogonalize newly computed Lanczos vectors against already converged Ritz vectors if necessary. The loss of orthogonality of the Lanczos vectors can cheaply be monitored from the eigenvectors of the small tridiagonal system. This is a reasonable approach because newly computed Lanczos vectors tend to diverge from the orthogonal direction into the direction of the converged eigenvectors. Another approach was proposed by Simon [Si84]. Here the loss of orthogonality is monitored by using a model of the process which causes the divergence from orthogonality. If a severe divergence is indicated by the model a newly computed Lanczos vector is orthogonalized against those Lanczos vectors it is tilting to. The latter method seems to be the more robust but both depend on an estimate of the error in performing the application of the operator to a vector. This is easily to obtain if this is only a vector by matrix multiply but harder if $\mathbf{A}$ is the operator from the next section.

chose r_0
repeat
$p_0 = Mr_0$
$\beta_0 = r_0 p_0$
chose shift σ
do j = 1, $n_{restart}$
$q_j = r_{j-1}\beta_{j-1}$
$p_j = p_{j-1}\beta_{j-1}$
$u_j = (K - \sigma M)^{-1} p_j$
$r_j = r_j - q_j \alpha_j$
$p_j = Mr_j$
$\beta_j = r_j p_j$
compute $\Theta_i, s_i, y_i, i = 1, \ldots, j$
orthogonalize r_j
enddo
chose new start vector

Box 2: Shift and Invert Lanczos

Shift and Invert

A remedy to the second problem with the simple algorithm is the use of a transformation of the original problem especially in the context of generalised eigenproblems $Kx = \lambda M$ (1).

Here the additional problem occurs if the mass matrix M is semi-definite only so the simple reduction the standard form is not possible by using a Cholesky decomposition of M. Instead of (1) the transformed equation

$$(K - \sigma M)^{-1} M x = \frac{1}{\lambda - \sigma} x \quad (2)$$

is used. It is easy to show that (λ, x) is an eigenpair of (1) if and only if $(1/(\lambda - \sigma), x)$ is an eigenpair of (2). This form has two advantages. Indefinite mass or stiffness matrices can be handled and using a proper shift σ the region of fast converging Ritz vectors can be moved through the spectrum to speed convergence. Especially the last fact makes the Lanczos algorithm so attractive. The price to pay for these advantages is the solution of a possibly indefinite linear system $(K - \sigma M)\, u_j = p_j$ (3).

This can be done by a symmetric decomposition $A = LDL^T$. The decomposition in addition provides invaluable information about the number of eigenvalues above and below σ by the Sylvester theorem. It states that the number of positive, negative and zero values, in the diagonal D of are the same as the number of eigenvalues below, above, and equal to σ, respectively. Using different shift requires the decomposition to be performed repetitively and to restart the Lanczos procedure. Because eigenpairs already calculated at previously chosen shifts should not be calculated again each newly computed Lanczos vector is besides some reorthogonalization scheme orthogonalized to all previously computed eigenvectors. The algorithm is given in box 2.

Problems in a Parallel Environment

Using parallel computers with distributed memories the two remarks in the last paragraph pose serious problems. Today no software for the direct solution of indefinite symmetric systems is in available on

these machines and no scalable algorithm has been designed yet. Even for positive definite matrixes of moderate size no scalable algorithm is available. While the solution of the linear systems can be performed by iterative methods, e. g. SYMMLQ or QMR [Bar94], there is no tool for checking the completeness of the calculated spectrum comparable to the robustness of the Sylvester theorem. The only fact one can rely on is the convergence property of Lanczos algorithm described above. In what follows large and small will refer to the eigenvalues of operator in (2). If the start vector of the Lanczos process is not by chance close to an eigenvector that corresponds to a small eigenvalue the process will make the Ritz values corresponding to the eigenvalues with the largest modulus converge first. One can be relatively sure that there are no further eigenvalues above the largest Ritz value if it has been accepted as a valid approximation to an eigenvalue. In [Gri94] a strategy for choosing the shift is described that is based on efficiency considerations only. In our implementation the shift strategy has to be adapted to the shortcoming of an algorithm not using the Sylvester theorem. If the shift is placed at a value much below the largest eigenvalue calculated so far there is a chance that some eigenvalues will be missed between the shift and this value. This may happen because the convergence rate is high only near the chosen shift but the strategy relies on the fact that the largest Ritz values will converge to the largest not calculated eigenvalue. If the shift is chosen somewhere inside the region of non accepted approximation a value may be accepted as the largest one in the current run which is not the largest because the convergence rate of the actually largest one is poor.

Shift Strategy in an Parallel Environment

In this section we shall describe the principal steps of our implementation of the Lanczos algorithm using a shift strategy. We shall use the term Lanczos run for a restarted Lanczos procedure with the same or a different shift (one step in the outer loop) and the term Lanczos step on iteration of the Lanczos algorithm (one step in the inner loop). Taking in account the observations described in the last paragraph the following strategy for choosing the shifts is adopted.

1) Chose a maximal number of Lanczos steps nrestart to be performed before the process is started.

2) Run the procedure until either nrestart iteration have been performed or the largest Ritz value in this run converges to an eigenvalue outside the interval we are interested in or there are no converging Ritz values among the Ritz values calculated in the current run. Here converging means that the estimated error of an approximation indicates that it will in a few iterations be accepted as a valid approximation to an eigenpair. This implies that the process is continued only as long as it seems to be useful to exploit the current shift.

3) Stop if the largest accepted approximation of an eigenvalue in this run is smaller than the lower bound of the interval of interest.

4) Chose a new starting vector as a linear combination of the unconverged Ritz vectors from the current run. Take only those which, although unconverged, have an error estimate indicating that the are an approximation to eigenvectors in the interval of interest. If there are no such vectors chose a random vector. The constructed vector is normalised and orthogonalized against all previously computed eigenvectors.

5) Chose a new shift if the Ritz pair corresponding to the largest Ritz value in the last run has converged. This means that there are no undetected eigenvalues above this one. The shift is then placed just above this value. If the largest Ritz value has not been accepted to be a valid approximation do not change the shift. Do not place it directly at this eigenvalue to prevent the operator from becoming singular. It is not placed below the largest eigenvalue to avoid the possibility of missing eigenvalues as explained above.

6) Go to 2)

Implementation

We implemented the algorithm described above using the PETSc library [Bal97] in the first place. This library is designed to be used in large scale scientific application. It provides the users with a high level view of the data. Although there is support for grid based calculation all computation can be performed in terms of matrices, vectors and index sets. Local and distributed data structures are not discriminated syntactically and no explicit parallel programming using MPI [MPI95] is necessary, usually. A task not tackled by this library yet is the problem of load balancing. PETSc is available on a wide range of machines: PC (running LINUX, FreeBSD, or NT4.0), all kinds of workstations, SMP (e. g. Dec or SGI), and MPP (e. g. Intel Paragon or Cray-T3E) multiprocessor machines. Although the program was written with parallel architectures in mind the development could be done almost entirely on a single processor workstation. MPI had to be employed directly only during the calculation of the data distribution which was primarily done by ParMetis [Kar95]. This program can be used to calculate a partition of a matrix for parallel computation by partitioning its graph. The resulting partition is optimized with respect to the number of edges cut by the separator and tries to keep the parts to be of almost equal size. This results in a close to optimal partition of data concerning load distribution and total amount of communication. The small triangular systems have been solved by routines from the LAPACK library [And95] and for the transformation of the eigenvectors from the small system s_i to the eigenvectors to the original system y the BLAS [Don88] routine for matrix-vector multiplication has been employed locally.

Examples

The following examples show the superior convergence of the algorithm if a shift strategy is used compared to the case without shift. The first example is taken from the Harwell-Boeing library [Duf89]. The first 20 eigenvalues are plotted in Figure 1. They contain some double and close triple eigenvalues. Table 1 shows the order the eigenvalues are extracted together with the shift chosen for each run. The shift does not change from run 3 to 4 and 5 to 6 because the largest eigenvalue converted in run 3 and 5 is almost the same as in run 4 and 6, respectively. This has to be compared to the calculation without shift laid out in Table 2. On a SGI Indigo2 workstation the whole calculation takes 22.8s using the shifting strategy and 90.7s without shift. For both cases a LU decomposition has been used to solve the linear systems

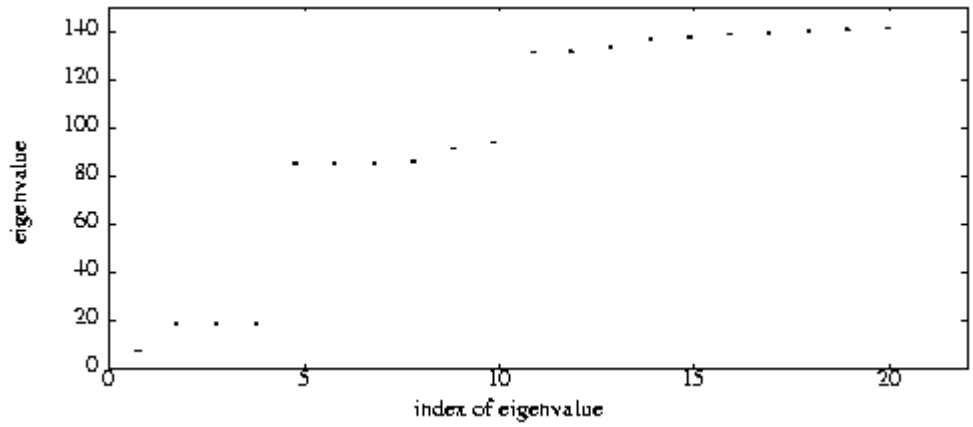

Figure 1: Eigenvalue distribution of bcsstk_08

run	1	2	3	4	5	6	7	8	9
eigenvalue									
6.9007	x								
18.1420		x							
18.1423		x							
18.1423			x						
84.7861				x					
84.7864				x					
84.7864					x				
85.5368				x					
91.0492				x					
93.4453				x					
130.9297						x			
131.4907						x			
132.9515						x			
136.2365						x			
137.2214						x			
138.4070							x		
138.7078							x		
139.5947							x		
140.5923							x		
141.0961								x	
shift	0	6.8	17.9	17.9	83.9	83.9	129.6	137.0	139.7
#steps	9	9	5	28	6	64	26	19	14

Table 1: The table shows the extraction of the eigenvalues using the shift strategy described in the last section. The top row indicates the number of the run while the bottom row gives the number of Lanczos steps in each run. The x marks the run in which an eigenvalue (given in the first column) is accepted. The total number of steps is 180

run	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
eigenvalue	x															
6.9007	x															
18.1420	x															
18.1423	x															
18.1423	x															
84.7861	x															
84.7864	x															
84.7864		x														
85.5368	x															
91.0492	x															
93.4453	x															
130.9297		x														
131.4907		x														
132.9515		x														
136.2365			x													
137.2214				x												
138.4070						x										
138.7078								x								
139.5947										x						
140.5923												x				
141.0961														x		
#steps	64	64	64	64	64	64	64	64	64	64	64	64	64	20	64	64

Table 2: For the same problem as in Table 1 the extraction of eigenvalues is shown for the case without shifting. Here the total number of steps is 992

# processors	total (sec)	set-up (sec)	solve (sec)	orthog (sec)	# iteration
2	875	43.7	816	6.3	2073
4	780	15.2	755	3.4	4370
8	676	5.8	663	1.7	8013
12	454	2.6	445	1.2	8264
16	397	1.8	390	0.9	9008
32	268	0.7	263	0.4	8184
64	228	0.3	224	0.2	8400

Table 3: Breakdown of total time of the main phases of the shift and invert algorithm. Set-up contains mainly the time for performing the local LU decomposition, solve the time for the solution of the system, and orthog the time for all orthogonalization required. The number of iteration gives the total number of iteration performed in the TFQMR algorithm

A larger example was solved on a Cray-T3E. The first 10 eigenvalues of the Laplacian on the unit square grid with 200x200 grid points with Dirichlet boundary conditions have been calculated. In the case with shift the transpose free QMR (TFQMR) method has been used for the solution of the linear

equations with an additive Schwarz preconditioner with overlap for preconditioning. The local systems were solved by LU decomposition and nrestart was set to 64. Using the shifting strategy it took 7 runs and a total of 78 steps to solve the problem. For this example the algorithm without using a shift failed to converge. A larger nrestart was required to extract all desired eigenpairs. Table 3 shows that the main bottleneck is the badly scaling solver while the set-up phase and the orthogonalization show good scaling. Apart from the usual increasing amount of communication relative to arithmetics this is due to an algorithmical fact. The quality of the chosen preconditioner decreases with increasing numbers of sub domains.

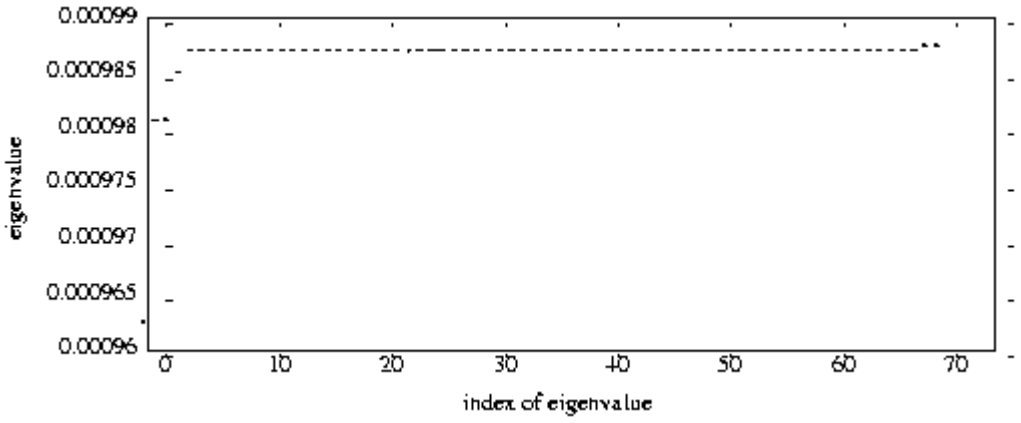

Figure 2: First 70 eigenvalues of BCSST_25

Finally, the ability of the code to reveal high multiplicity was check using the problem BCSSTK_25 from the Harwell-Boeing collection. The distribution of the first 70 eigenvalues is given in Figure 2. There is a multiplicity of 64 for the fifth eigenvalue. The program was run with nrestart=128 and it took 6 runs and a total of 416 Lanczos steps to reveal all the 70 eigenvalues including the very high multiplicity. The first 4 and the last 2 eigenvalues together with 4 of the 64 fold ones are extracted at the first run. It took 4 more runs to reveal all the 64 fold ones and a final run to make sure that there are no undetected eigenvalues inside the interval of interest. The whole program took 7141sec on 32 processors of a T3E. 99.4% of the total time were spent in solving the linear system. These shows that using the shift strategy can handle even this very difficult case despite the potentially unsafe algorithm due to the lack of checking the completeness of the calculated spectrum rigorously. Furthermore, a better solver is required to obtain a faster solution of the linear system.

To improve the performance of this algorithm an extension to the block algorithm as described in [Gri94] is under development which should give a higher convergence rate for the Lanczos process. Additionally it should result in a better performance of the floating point operation and decrease the communication overhead due to the possibility of blocking.

Parallel Direct Solvers

The timings in the previous paragraph clearly show that the bottleneck in the solution of the eigenproblem is the solution of the linear systems. Instead of using an iterative solver it may be possible to use a direct one. The question arises if it can be competitive to an iterative one when solving eigenvalue problems.

A direct solver works by calculating a decomposition of the system matrix $A = LU$ where L and U are lower and upper triangular matrices, respectively. Than the system $Ax = b$ is solved by calculating $y = L^{-1}b$ and $x = U^{-1}b$. The algorithm has three principal phases: the ordering of the matrix, the factorization, and the solution of triangular systems. In the ordering phase the matrix is reorder to minimize the size of the factors of the decomposition calculated in the factorization phase. Eventually, the solution is calculated by solving the two triangular systems. What makes this procedure attractive in

the context of the Lanczos algorithm is the fact that the systems are solved repetitively for different right hand sides, i. e. a once computed factorization can be used for many solves. This triangular solve is by far the least time consuming part of a direct solver. Here we shall review some publicly available direct solver and give some performance results and lay out the main ideas of an implementation by ourselves.

Current State of Available Software

The first two solvers described serve for the solution of positive definite symmetric matrices. This restriction simplifies the development greatly because no pivoting has to be performed and a symmetric decomposition $A = LL^T$ is possible. This allows to split the decomposition of the matrix into a numeric and symbolic phase. During the symbolic phase only index information is required and the data structure for the numerical phase can be created. These two phases have to be interweave if pivoting is necessary as in the case of indefinite or unsymmetric systems. Pivoting results, beside complicating the process of factorization, in global communication and synchronization thereby degrading the performance of the algorithm.

The CAPSS solver [Hea97] uses Cartesian nested dissection to partition and order the matrix. Therefore coordinates of the grid points are required. An implementation for an Intel Paragon only is available. It is rather a research code to show the feasibility of parallelizing all parts of the direct solution of a linear system put into the public domain than a code that can be used as a reliable solver in real application. Although every single step is parallelized the whole program shows only very limited speedup but there are many parts which can be improved.

The WSSMP solver of Gupta [Gup97] is designed for more general positive definite systems, i. e. no coordinate information is required. It is available on IBM SP2 machines only although it was originally developed on a Cray-T3D. The symbolic factorization is performed sequentially only and the ordering phase is only partially parallelized but it is robust and supported software.

SuperLU [Li96] is designed for unsymmetric general matrices. Ordering for stable pivoting is an essential part of this software. Up to now only a multiprocessor shared memory version for the decomposition phase is available. It aims to give reasonable speedup for a moderate number of processors. Despite all these drawbacks it is the only available and at least partially parallelized direct solver for indefinite matrices.

The PARASOL [Par96] project up to now has, important enough, only delivered the definition of an interface which should allow to call different solvers by the same calling sequence. Available are only a test driver which calls a solver and preconditioner from the PIM library. Direct solvers are in the state of prototype implementation but not available yet.

Performance Tests

With two of the above mentioned software packages (WSSMP and SuperLU) we performed performance studies with problems supplied by the McNeal-Schwendler Corporation. WSSMP has been run on an IBM SP2 with 8 nodes each with 128MB memory (Thin2-nodes). For comparison these problems have been solved using the SuperLU package as well.

	N	#NNZ(A)	#NNZ(L)
bst10a	515	9273	11852
bst30a	4515	99607	223767
bst120a	53570	1174418	4657521
xcarbody	47072	2336898	6334584

Table 4: Size of problems supplied by MSC. N is the order of the matrix, #NNZ(A) the number of non zero entries in the matrix and #NNZ(L) the number of nonzero entries in one of the triangular factors

#PROCS	ORDER	SYM	NUM	SOLVE	TOTAL
1	0.08	0.01	0.01	0.00	0.11
2	0.09	0.02	0.02	0.01	0.15
4	0.08	0.04	0.03	0.02	0.16
8	0.11	0.05	0.05	0.03	0.23

Table 5: Times in seconds for different phases in WSSP on a SP2 for example bst10a

#PROCS	ORDER	SYM	NUM	SOLVE	TOTAL
1	0.583	0.117	0.204	0.028	0.932
2	0.422	0.165	0.222	0.037	0.846
4	0.350	0.200	0.194	0.044	0.788
8	0.362	0.222	0.186	0.057	8.827

Table 6: Times in seconds for different phases in WSSP on a SP2 for example bst30a

#PROCS	ORDER	SYM	NUM	SOLVE	TOTAL
1	12.80	1.48	6.05	0.62	20.95
2	10.70	1.86	4.51	0.38	17.45
4	7.21	2.14	2.98	0.26	12.59
8	5.98	2.63	1.83	0.18	10.62

Table 7: Times in seconds for different phases in WSSP on a SP2 for example bst120a

#PROCS	ORDER	SYM	NUM	SOLVE	TOTAL
1	5.24	2.66	21.73	2.65	32.28
2	4.33	2.81	6.50	0.31	13.95
4	4.06	3.10	4.25	0.22	11.63
8	3.90	3.27	3.03	0.17	10.37

Table 8: Times in seconds for different phases in WSSP on a SP2 for example xcarbody

	ORDER	FACTOR		SOLVE		MEMORY	PROCESSOR TYPE	TOTA
	sec	sec	MFlop/s	sec	MFlop/s	MB		sec
bst10a	0.07	0.06	9.8	0.01	5.4	0.38	590H	0.14
bst30a	0.77	1.82	16.9	0.06	7.4	8.13	590H	2.65
bst120a	6.67	303.45	20.0	5.22	6.8	193.99	R30	315.44
xcarbody	29.53	745.48	19.4	10.15	5.6	304.07	R30	786.16

**Table 9: Times and performance for various phases of the SuperLU solver on an SP2.
Processor type 590H is equipped with 128MB main memory only while the R30 processors have 512MB.
The floating point performance of the 590H is about 2.5 better than those of the R30**

Table 5 to Table 8 show the timings for the different phases of the solution process: ordering (ORDER), symbolical factorization (SYM), numerical factorization (NUM), triangular solve (SOLVE), and the total time for the solution (TOTAL). Table 5 and Table 6 show that for relative small problems no efficient use of more than one processor is possible. Only for moderately sized problems like bst120a and xcarbody with about 50000 unknowns a considerable speedup is observed. The decrease of computation time for the example xcarbody from one to two processors is mainly due to the fact that on one processors the problem does not fit completely into memory and is partially put out to disk by the operating system. So for both larger problems the speedup on 8 processors is about four for the completely parallelized phases of numerical factorization and triangular solve. This indicates that not much can be gained when using a much higher number of processors for this size of problems. On the other hand the times for ordering and symbolic factorization show the urgent need for parallelizing these parts as well. Already on two nodes they are these two phases dominate the total time for the solution of one system. The problem becomes less dominant when multiple right hand sides have to be solves for the same matrix.

The timings in Table 9 for the SuperLU program show that it is not competitive to WSSMP in solving positive definite systems both in memory requirements and computation time. This is entirely because it is designed for the more complex problem of solving general unsymmetric systems. The larger example had to be run on a slower machine which was equipped with sufficient memory. We started to port the program to a NEC-SX4 where we achieved (on a single processor) 120 Mflop/s for bst120a and 250 Mflop/s for an unsymmetric matrix of order 26000 with about 117000 non zero entries. This is far from the peak performance of 2Gflop/s which is mainly due to problems of the C compiler in vectorizing some complex loops. However, results in [Li96] show the good performance of the code over a wide range of other multiprocessor architectures.

New Implementation

The summary above showed despite software for positive definite system is available these implementations focus on different parts of the algorithm and are available only on a limit number of architectures. In the WSSMP solver for example a significant part of the algorithm is executed on one processors only which clearly impedes an overall scalebility. The CAPSS solver is available only for an Intel Paragon because it uses the native message passing library. On the other hand much work has been done on the various phases, see e. g. [Liu92] and the references therein. This work can be used as building blocks for a new implementation of a parallel direct solver. We aim at a solver for distributed memory architecture based on the MPI message passing standard and using libraries wherever possible. In this section we shall shortly review the essential ingredients of the algorithm.

The first part that has been neglected so far is the initial data distribution which has to be at least partially parallelized to not dominate the total solution time for large problems. Here it should be possible to use the new MPI-2 I/O routines as specified in [MPI97] which will be available soon on all

architectures supporting the MPI standard. Today only some of the routines are available [Tha97] but they should suffice to tackle this task. The algorithm for the solution itself is based on a domain decomposition approach similar to the one used in CAPSS but based on the graph of the matrix instead on the partitioning of the grid. A tool for this task is the ParMetis program a parallel version of the program Metis described in [Kar95]. For the symbolic factorization phase based on this decomposition a scalable algorithm is described in [Hea93]. The numerical factorization will follow the algorithm given in [Liu92] based on the multifrontal approach of Duff and Reid [Duf83]. The final triangular solve phase will be implemented as described in [Gup95] and [Hea88].

Iterative Solver

For comparison we solved the problems of the last section using the best iterative solvers from the PETSc library. Here the CG method in conjunction with an overlapping additive Schwarz preconditioner is used. The local systems are solved by LU decomposition.

The optimal overlap depends on the size of the problem. A large overlap increases the convergence rate for the price of a larger number of operation and a larger amount of communication. For larger problems this overhead is outweighed by the increase in convergence speed while for smaller ones it does not pay to use a large overlap.

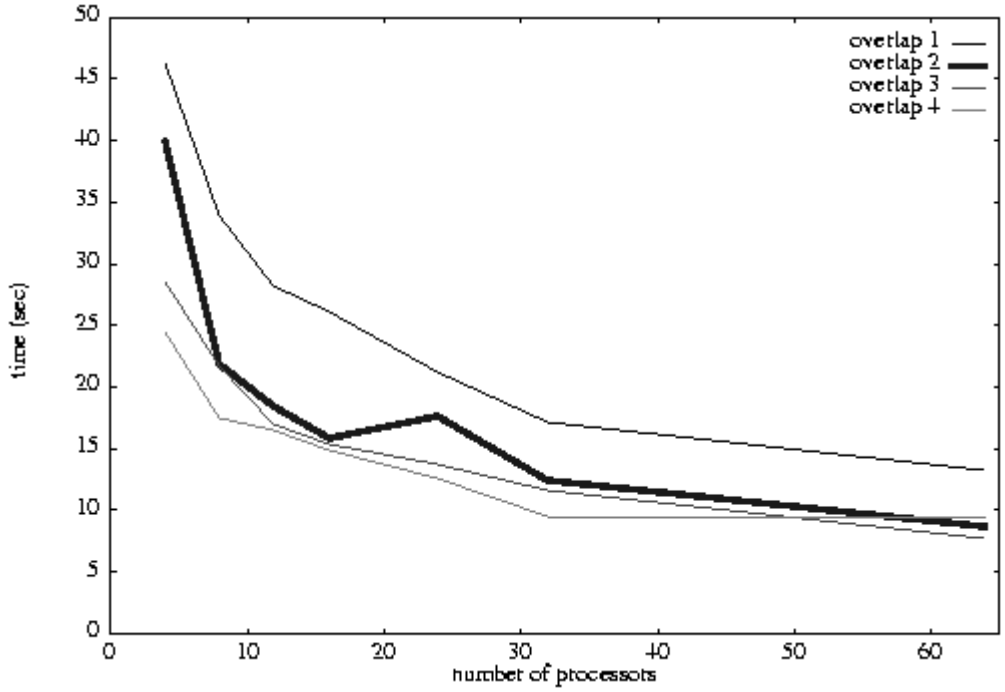

Figure 3: Run time for iterative solution of bst120a with different amount of overlap in the preconditioner. These results have been obtain with a slightly different setting of parameters as for the following figures

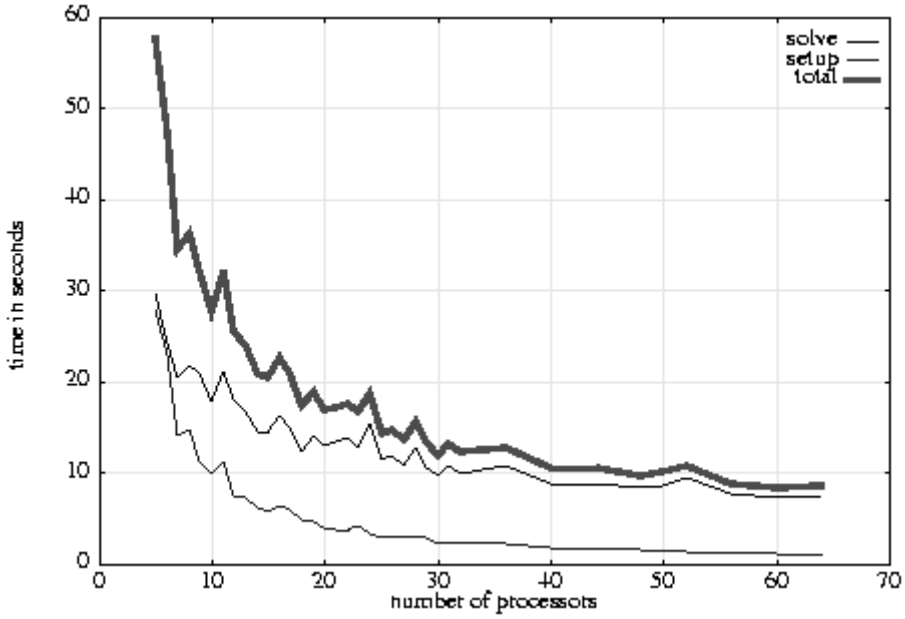

Figure 4: Breakdown of total time into setup and solution phase for bst120a using an overlap of 3

As Figure 3 shows, only if the size of the local problem is large enough, i. e. for a small number of nodes, it is efficient to use a large overlap. The best choice of the amount of overlap is a delicate task because it depends on many parameters. The gain in convergence rate has to overbalance the additional amount of communication and arithmetic operation incurred by extending the overlap. While the additional overhead in arithmetics and communication can in principle be specified it is in general impossible to quantify the increase in the convergence rate. Therefore determining the size of the overlap has to be chosen by experiments. Aside from speeding up the convergence a larger overlap increases the robustness of the solver.

Figure 4 shows the breakdown of the run time into setup and solve phase for the iterative solution of the problem bst120a. The setup phase essentially consists of calculating the LU decomposition of the local part of the matrix on each processor and the solve phase of executing the CG algorithm with preconditioning. It can be seen easily that for the domain decomposition preconditioner used the time for the setup of the solver decreases rapidly due to the decreasing size of the local systems. On the other hand, the time for the solution decreases only slowly with the number of processors involved. The main reason is the dependence of the quality of the preconditioner on the number of subdomains. The number of iteration required to reach the termination criterion increases with the number of processors. Figure 5 shows an almost linear increase of the number of iteration required with the number of processors. The total speedup on 64 processors amounts to about 33 if an overlap of three is used giving an efficiency of 53%. The speedup is measured relative to the run with five processors, i. e. the speedup on n processors is given by $5 \times t_5 / t_n$. Because of the ever decreasing efficiency of the preconditioner it is not to be expected to gain much with even higher processor numbers. For 128 processors the total time decreases to only 75% of the time for 64 processors, i. e. the efficiency now is only 35%.

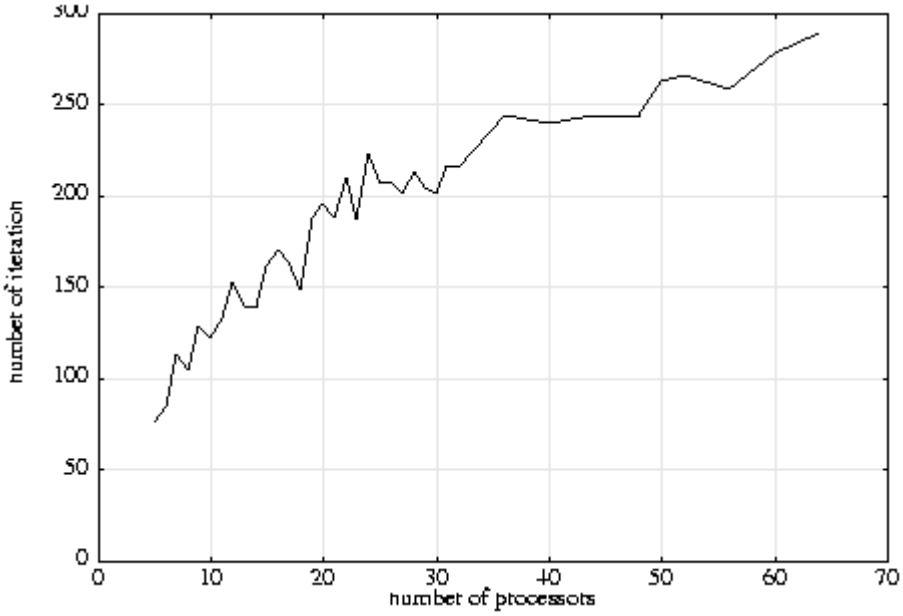

Figure 5: Number of iterations for example bst120a for different number of processors using an overlap of 3

problem	#processors	overlap	setup	solve	total
bst120a	4	4	31.84	24.33	56.17
bst120a	8	4	13.08	17.41	30.49
bst120a	32	4	2.79	9.35	12.14
bst120a	64	3	1.05	7.45	8.50
xcarbody	8	2	16.68	57.16	72.84
xcarbody	32	2	4.29	36.05	39.34
xcarbody	64	1	1.13	27.96	29.09

Table 10: Times for the iterative solution of the two larger problems for a selected number of processors

For comparison to the direct solver results Table 10 lists the times for the two larger examples. The total solution time on a single processor for bst120a with WSSMP was 21s and 32s for xcarbody. In the first case about 12 processors of a T3E have to be used to achieve the solution in about the same time while for the second about 60 processors are needed. This does not necessarily mean that iterative solver are inferior to direct solvers because there is still room for improvement but it highlights the higher robustness of direct solvers. Especially in the finite element context much can be gained using a more elaborated preconditioner. For example if the unassembled matrix is available it is possible to construct an efficient preconditioner based on the element matrices [Day95]. Furthermore, for much larger problems the iterative solver will eventually be faster due to better asymptotic arithmetic complexity of the conjugate gradient method and consume less memory than direct methods.

No final judgement can be made which of them is the more useful in solving eigenvalue problems on parallel architectures. Although on a small number of processors and for the size of problems we investigated the direct solver is much faster than the iterative one they are not as scalable and require a much larger amount of memory so it may be that the savings by reusing a once calculated decomposition may not pay off. Furthermore because no software for the parallel solution of indefinite systems is available so far a shift strategy can not be used resulting in a severe degradation of the performance of the Lanczos algorithm.

Bibliography

[And95] Anderson, E., Z . Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users Guide, SIAM, Philadelphia, USA

[Don88] An extended set of FORTRAN Basic Linear Algebra Subprograms, ACM Trans. Math. Soft., 14, pp. 1-17, 1988

[Bal97] Balay, S., W. Gropp, L. C. McInnes, B. Smith, PETSc 2.0 Users Manual, Technical Report ANL-95-11, Revision 2.0.17, Argonne National Laboratory, Argonne, USA, 1997

[Bar94] Barrett, V., Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA, USA, 1994

[Day95] Daydé, M. J., J.-Y. L'Excellent, N. I. M. Gould, Solution of structured systems of linear equations using element-by-element preconditioners, in 2nd IMACS Symposium on iterative methods in linear algebra, Blagoevgrad, Bulgaria, 1995

[Duf83] Duff, I. S., J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Software 9, pp 302-325, 1983

[Duf89] Duff, I. S., R. Grimes, J. Lewis, Sparse matrix test problems. ACM Trans. Mathematical Software, 15, pp. 1-14, 1989

[Gri94] Grimes, R. G., J. G. Lewis, H. D. Simon, A shifted block Lanczos algorithm for solving sparse symmetric generalised eigenproblems, SIAM J. Mat. Anal. Appl., 15, pp. 228-272, 1994

[Gup95] Gupta, A., V. Kumar, Parallel Algorithms for forward and backward substitution in direct solution of sparse linear systems, Technical Report, University of Minnesota, Department of Computer Science, Minneapolis, USA, 1995

[Gup97] Gupta, A., G. Karypis, V. Kumar, Highly scalable parallel algorithms for sparse matrix factorization, IEEE Transaction on Parallel and Distributed Systems, 8, 1997

[Hea88] Heath, M., C. Romine, Parallel solution of triangular systems on distributed memory multiprocessors, SIAM J. Sci. Stat. Comput., 9, pp 558-588, 1988

[Hea93] Heath, M. T., P. Raghavan, Distributed Solution of sparse linear systems, Lapack working note 62, UT-CS-93-201, Technical Report, University of Tennessee, Knoxville, 1993

[Hea97] Heath, M. T., P. Raghavan, Performance of a fully parallel sparse solver, The International Journal of Supercomputing Applications and High Performance Computing, 11, pp. 49-64, 1997

[Kar95] Karypis, G., V. Kumar, METIS, Unstructured Graph Partitioning and Sparse Matrix Ordering System, University of Minnesota, Technical Report, Department of Computer Science, Minneapolis, MN, USA, 1995

[Li96] Li, X. S., Sparse Gaussian Elimination on High Performance Computers, PhD Thesis, University of California, Berkley, 1996

[Liu92] Liu, J. W.-H., The multifrontal method for sparse matrix solution: Theory and practice, SIAM Review, 34, pp. 82-109, 1992

[MPI95] MPI: MPI: A Message-Passing Interface Standard, University of Tennessee, Knoxville, Tennessee, 1995

[MPI97] MPI-2: Extensions to the Message-Passing Interface, University of Tennessee, Knoxville, Tennessee, 1997

[Nou87] Nour-Omid, B., B. N. Parlett, T. Ericsson, P. S. Jensen, How to implement the spectral transformation, Math. Comp., 48, pp. 663-673, 1987

[Pai72] Paige, C. C., Computational variants of the Lanczos method for the eigenproblem, J. Inst.

Math. Appl., 10, pp. 373-381, 1972

[Par96]

[Par79] Parlett, B. N., D. S. Scott, The Lanczos Algorithm with selective orthogonalization, Math. Comp., 33, pp. 217-238, 1979

[Par80] Parlett, B. N., The symmetric eigenvalue problem, Prentice-Hall, 1980

[Sim84] Simon, H. D., The Lanczos Algorithm with partial reorthogonalization, Math. Comp., 42, pp.115-142, 1984

[Tha97] Thakur, R., E. Lusk, W. Gropp, Users guide for ROMIO, ANL/MSC-TM-234, Argonne National Laboratory, Argonne, Illinois, USA, 1997

Dr. Hans-Peter Kersken, NA-5784
E-Mail: kersken@rus.uni-stuttgart.de

Uwe Küster, NA-5984
E-Mail: küster@hlrs.de