Parallel Numerics '05, 25-34 M. Vajteršic, R. Trobec, P. Zinterhof, A. Uhl (Eds.)
Chapter 2: Matrix Algebra ISBN 961-6303-67-8

Parallel Maxwell Eigensolver Using Trilinos Software Framework *

Peter Arbenz [1], Martin Bečka [2,†], Roman Geus [3], Ulrich Hetmaniuk [4], Tiziano Mengotti [1]

[1] Institute of Computational Science, Swiss Federal Institute of Technology, CH-8092 Zurich, Switzerland

[2] Department of Informatics, Institute of Mathematics, Slovak Academy of Sciences, Dúbravská cesta 9, 841 04 Bratislava, Slovak Republic

[3] Paul Scherrer Institut, CH-5232 Villigen PSI, Switzerland

[4] Sandia National Laboratories, Albuquerque, NM 87185-1110, U.S.A. ‡

We report on a parallel implementation of the Jacobi–Davidson algorithm to compute a few eigenvalues and corresponding eigenvectors of a large real symmetric generalized matrix eigenvalue problem. The eigenvalue problem stems from the design of cavities of particle accelerators. It is obtained by the finite element discretization of the time-harmonic Maxwell equation in weak form by a combination of Nédélec (edge) and Lagrange (node) elements. We found the Jacobi–Davidson (JD) method to be a very effective solver provided that a good preconditioner is available for the correction equations. The parallel code makes extensive use of the Trilinos software framework. In our examples from accelerator physics we observe satisfactory speedup and efficiency.

†Corresponding author. E-mail: martin.becka@savba.sk

‡Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

1 Introduction

Many applications in electromagnetics require the computation of some of the eigenpairs of the curl-curl operator,

$$\mathbf{curl}\,\mu_r^{-1}\mathbf{curl}\,\mathbf{e}(\mathbf{x}) - k_0^2\,\varepsilon_r\,\mathbf{e}(\mathbf{x}) = \mathbf{0}, \qquad \operatorname{div}\mathbf{e}(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Omega, \tag{1}$$

Equations (1) are obtained from the Maxwell equations after separation of the time and space variables and after elimination of the magnetic field intensity. The discretization of (1) by finite elements leads to a real symmetric generalized matrix eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad C^T\mathbf{x} = \mathbf{0}, \tag{2}$$

where A is positive semidefinite and M is positive definite. In order to avoid spurious modes we approximate the electric field $\mathbf{e}$ by Nédélec (or edge) elements [17]. The Lagrange multiplier that is a function introduced to treat properly the divergence free condition is approximated by Lagrange (or nodal) finite elements [3].

In this paper we consider a parallel eigensolver for computing a few of the smallest eigenvalues and corresponding eigenvectors of (2) as efficiently as possible with regard to execution time and memory cost. In earlier studies [3] we found the Jacobi–Davidson algorithm [18, 9] a very effective solver for this task. We have parallelized this solver in the framework of the Trilinos parallel solver environment [10].

In section 2 we briefly review the symmetric Jacobi-Davidson eigensolver and the preconditioner that is needed for its efficient application. In section 3 we discuss data distribution and issues involving the use of Trilinos.

In section 4 we report on experiments that we conducted by means of problems originating in the design of the RF cavity of the 590 MeV ring cyclotron installed at the Paul Scherrer Institute (PSI) in Villigen, Switzerland. These experiments indicate that the implemented solution procedure is almost optimal in that the number of iteration steps until convergence only slightly depends on the problem size.

2 The eigensolver

The Jacobi–Davidson algorithm has been introduced by Sleijpen and van der Vorst [18]. There are variants for all types of eigenvalue problems [5]. Here we use a variant (JDSYM) adapted to the generalized symmetric eigenvalue problem (2) as described in detail in [2, 9]. This algorithm is well-suited since it does not require the factorization of the matrices A or M. In [2, 3, 4] we found JDSYM to be the method of choice for this problem.

In addition to the standard JDSYM algorithm, we keep solutions of the correction equation orthogonal to C applying a projector operator $(I - YH^{-1}C^T)$ in each iteration step. Note that $Y = M^{-1}C$ is a very sparse basis of the null space of A and that $H = Y^TC$ is the discretization of the Laplace operator in the nodal element space [3].

Our preconditioner $K \approx A - \sigma M$, where σ is a fixed shift, is a combination of a hierarchical basis preconditioner and an algebraic multigrid (AMG) preconditioner.

Since our finite element spaces consist of Nédélec and Lagrange finite elements of degree 2 and since we are using hierarchical bases, we employ the hierarchical basis preconditioner that we used successfully in [3]. Numbering the linear before the quadratic degrees of freedom, the matrices A, M and K get a 2-by-2 block structure. The $(1, 1)$-blocks correspond to the bilinear forms involving linear basis functions. The hierarchical basis preconditioners as discussed by Bank [6] are stationary iteration methods that respect the 2-by-2 block structure of A and M. We use the symmetric block Gauss–Seidel iteration as the underlying stationary method.

For very large problems (order 10^5 and more), we solve with K_{11} by a single V-cycle of an AMG preconditioner. This makes our preconditioner a true multilevel preconditioner. We found ML [16] the AMG solver of choice as it can handle unstructured systems that originate from the Maxwell equation discretized by linear Nédélec finite elements. ML implements a smoothed aggregation AMG method [20] that extends the straightforward aggregation approach of Reitzinger and Schöberl [14]. ML is part of Trilinos which is discussed in the next section.

The approximation $\widetilde{K}_{22}$ of K_{22} again represents a stationary iteration method of which we execute a single iteration step.

3 Parallelization issues

For very large problems, the data must be distributed over a series of processors. To make the solution of these large problems feasible, an efficient parallel implementation of the algorithm is necessary. Such a parallelization of the algorithm requires proper data structures and data layout, some parallel direct and iterative solvers, and some parallel preconditioners. For our project, we found the Trilinos Project [19] to be an efficient environment to develop such a complex parallel application.

3.1 Trilinos

The Trilinos Project is an ongoing effort to design, develop, and integrate parallel algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications [19, 10, 15]. Trilinos is a collection of compatible software packages. Their capabilities include parallel linear algebra computations, parallel algebraic preconditioners, the solution of linear and non-linear equations, the solution of eigenvalue problems, and related capabilities. Trilinos is primarily written in C++ and provides interfaces to essential Fortran and C libraries.

For our project, we use the following packages

- Epetra, the fundamental package for basic parallel algebraic operations. It provides a common infrastructure to the higher level packages,
- Amesos, the Trilinos wrapper for linear direct solvers (SuperLU, UMFPACK, KLU, etc.),
- AztecOO, an object-oriented descendant of the Aztec library of parallel iterative solvers and preconditioners,
- ML, the multilevel preconditioner package, that implements a smoothed aggregation AMG preconditioner capable of handling Maxwell equations [7, 16].

For a detailed overview of Trilinos and its packages, we refer the reader to [10].

3.2 Data structures

Real valued double precision distributed vectors, multivectors (collections of one or more vectors) and (sparse) matrices are fundamental data structures, which are implemented in Epetra. The distribution of the data is done by specifying a communicator and a map, both Epetra objects.

The notion of a communicator is known from MPI [13]. A communicator defines a context of communication, a group of processes and their topology, and it provides the scope for all communication operations. Epetra implements communicators for serial and MPI use. Moreover, communicator classes provide methods similar to other MPI functions.

Vectors, multivectors and matrices are distributed row wise. The distribution is defined by means of a map. A map can be defined as the distribution of a set of integers across the processes, it relates the global and local row indices. To create a map object, a communicator, the global and local numbers of elements (rows), and the global numbering of all local elements have to be provided. So, a map completely describes the distribution of vector elements or matrix rows. Note that rows can be stored on several processors redundantly. To create a distributed vector object, in addition to a map, one must assign values to the vector elements. The Epetra vector class offers standard functions for doing this and other common vector manipulations.

Trilinos supports dense and sparse matrices. Sparse matrices are stored locally in the compressed row storage (CRS) format [5]. Construction of a matrix is row by row or element by element. Afterwards, a transformation of the matrix is required in order to perform matrix-(multi)vector product $Y = A \times X$ efficiently, specifying maps of the vectors X and Y.

Some algorithms require only the application of a linear operator, such that the underlying matrix need not be available as an object. Epetra handles this by means of a virtual operator class. Epetra also admits to work with block sparse matrices. Unfortunately, there is no particular support for symmetric matrices.

To *redistribute* data, one defines a new, so-called target map and creates an empty data object according to this new map as well as an Epetra's import/export object from the original and the new map. The new data object can be filled with the values of the original data object using the import/export object, which describes the communication plan.

3.3 Data distribution

A suitable data distribution can reduce communication costs and balance the computational load. The gain from such a redistribution can, in general, overcome the cost of this preprocessing step.

Zoltan [21, 8] is a library that contains tools for load balancing and parallel data management. It provides a common interface to graph partitioners like METIS and ParMetis [12, 11]. Zoltan is not a Trilinos package. But the Trilinos package EpetraExt provides an interface between Epetra and Zoltan.

In our experiments, we use ParMetis to distribute the data. This partitioner tries to distribute a graph such that (I) the number of graph vertices per processor is balanced and (II) the number of edge cuts is minimized. The former balances the work load. The latter minimizes the communication overhead by concentrating elements in diagonal blocks and minimizing the number of non-zero off-diagonal blocks. In our experiments, we define a graph G, which contains connectivity informations for each node, edge, and face of the finite element mesh. G is constructed from portions of the sparse matrices M, H, and C. To reduce overhead of the redistribution we also work with a smaller than our artificial graph G. We determine a good parallel distribution just for the vertices and then adjust the edges and faces accordingly.

4 Numerical experiments

In this section, we discuss the numerical experiments used to assess the parallel implementation. Results have been presented in [1].

The experiments have been executed on a 32 dual-node PC cluster in dedicated mode. Each node has 2 AMD Athlon 1.4 GHz processors, 2 GB main memory, and 160 GB local disk. The nodes are connected by a Myrinet providing a communication bandwidth of 2000 Mbit/s. The system operates with Linux.

For these experiments, we use the developer version of Trilinos on top of MPICH. We computed the 5 smallest positive eigenvalues and corresponding eigenvectors using JDSYM with the multilevel preconditioner.

Test problems originate in the design of the RF cavity of the 590 MeV ring cyclotron installed at the Paul Scherrer Institute (PSI) in Villigen, Switzerland. We deal with two problem sizes. They are labelled `cop40k` and `cop300k`. Their characteristics are given in Table 1, where we list the order n and the

Table 1: Matrix characteristics

grid	$n_{A-\sigma M}$	$nnz_{A-\sigma M}$	n_H	nnz_H
cop40k	231,668	4,811,786	46,288	1,163,834
cop300k	1,822,854	39,298,588	373,990	10,098,456

number of non-zeros nnz for the shifted operator $A - \sigma M$ and for the discrete Laplacian H. Here the eigenvalues to be computed are

$$\lambda_1 \approx 1.13, \quad \lambda_2 \approx 4.05, \quad \lambda_3 \approx 9.89, \quad \lambda_4 \approx 11.3, \quad \lambda_5 \approx 14.2.$$

We set $\sigma = 1.5$.

In Table 2, we report the execution times $t = t(p)$ for solving the eigenvalue problem with various numbers p of processors. These times do not include preparatory work, such as the assembly of matrices or the data redistribution. $E(p)$ describes the parallel efficiency with respect to the simulation run with the smallest number of processors. t_{prec} and t_{proj} indicate the percentage of the time the solver spent applying the preconditioner and the projector, respectively. $n_{\text{inner}}^{\text{avg}}$ is the average number of inner (QMRS) iterations per outer iteration. The total number of applications for the preconditioner K is approximately $n_{\text{outer}} \cdot n_{\text{inner}}^{\text{avg}}$. Here we use an AMG preconditioner for the block K_{11}, Jacobi steps for K_{22} and an AMG preconditioner for the whole H.

In Table 3, we use the AMG preconditioner for the block K_{11}, Jacobi steps for K_{22}, and a similar strategy for H (AMG preconditioner for H_{11} and Jacobi steps for H_{22}). We investigate the effect of redistributing the matrices.

Table 2: Results for matrix `cop40k`

p	t [sec]	$E(p)$	t_{prec} [%]	t_{proj} [%]	n_{outer}	$n_{\text{inner}}^{\text{avg}}$
1	2092	1.00	37	18	53	19.02
2	1219	0.86	38	17	54	18.96
4	642	0.81	37	17	54	19.43
8	321	0.81	38	18	53	19.23
12	227	0.77	40	19	53	19.47
16	174	0.75	43	20	53	18.96

Results in Table 3 show that the quality of data distribution is important. For the largest number of processors ($p = 16$), the execution time with the redistributed matrices is half the time obtained with the original matrices. These were straightforward block distributions of the matrices.

Table 3: `cop40k`: Comparison of results with (left) and without (right) redistribution

p	t [sec]		$E(p)$		n_{outer}		$n_{\text{inner}}^{\text{avg}}$	
1	1957	2005	1.00	1.00	53	53	19.02	19.02
2	1159	1297	0.84	0.77	54	53	19.06	19.66
4	622	845	0.79	0.59	54	55	19.43	19.18
8	318	549	0.77	0.45	53	54	19.23	19.67
12	231	451	0.71	0.37	53	54	20.47	19.78
16	184	366	0.66	0.34	53	54	19.00	19.04

Finally, in Table 4, we report results for our larger problem size `cop300k`. We use the 2-level preconditioner for K and H: an appropriate AMG preconditioner for the blocks K_{11} and H_{11} and one step of Jacobi for the blocks K_{22} and H_{22}. Table 4 shows that, for these experiments, the iteration counts behave nicely and that efficiencies stay high.

Table 4: Results for matrix `cop300k`

p	t [sec]	$E(p)$	n_{outer}	$n_{\text{inner}}^{\text{avg}}$
8	4346	1.00	62	28.42
12	3160	0.91	62	28.23
16	2370	0.92	61	28.52

5 Conclusions

In conclusion, the parallel algorithm shows a very satisfactory behavior. The efficiency of the parallelized code does not get below 65 percent for 16 processors. We usually have a big efficiency loss initially. Then efficiency decreases slowly as the number of processors increases. This is natural due to the growing communication-to-computation ratio.

The accuracy of the results are satisfactory. The computed eigenvectors were M-orthogonal and orthogonal to C to machine precision.

We plan to compare the Jacobi-Davidson approach with another eigenvalue solvers like the locally optimal block preconditioned conjugate gradient methods (LOBPCG).

References

[1] P. Arbenz, M. Bečka, R. Geus, U. Hetmaniuk, and T. Mengotti, On a Parallel Multilevel Preconditioned Maxwell Eigensolver. *Technical Report 465*, Institute of Computational Science, ETH Zürich, December 2004.

[2] P. Arbenz and R. Geus, A comparison of solvers for large eigenvalue problems originating from Maxwell's equations. *Numer. Linear Algebra Appl.*, 6(1):3–16, 1999.

[3] P. Arbenz and R. Geus, Multilevel preconditioners for solving eigenvalue problems occuring in the design of resonant cavities. *Applied Numerical Mathematics*, 2004. Article in press. Corrected proof available from `doi:10.1016/j.apnum.2004.09.026`.

[4] P. Arbenz, R. Geus, and S. Adam, Solving Maxwell eigenvalue problems for accelerating cavities. *Phys. Rev. ST Accel. Beams*, 4:022001, 2001. (Electronic journal available from `http://prst-ab.aps.org/`).

[5] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* SIAM, Philadelphia, PA, 2000.

[6] R. E. Bank, Hierarchical bases and the finite element method. *Acta Numerica*, 5:1–43, 1996.

[7] P. B. Bochev, C. J. Garasi, J. J. Hu, A. C. Robinson, and R. S. Tuminaro, An improved algebraic multigrid method for solving Maxwell's equations. *SIAM J. Sci. Comput.*, 25(2):623–642, 2003.

[8] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[9] R. Geus, *The Jacobi–Davidson algorithm for solving large sparse symmetric eigenvalue problems.* PhD Thesis No. 14734, ETH Zürich, 2002. (Available at URL `http://e-collection.ethbib.ethz.ch/show?type=diss&nr=14734`).

[10] M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, An overview of the Trilinos Project. *ACM Trans. Math. Softw.*, 5:1–23, 2003.

[11] G. Karypis and V. Kumar, Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Rev.*, 41(2):278–300, 1999.

[12] METIS: A family of programs for partitioning unstructured graphs and hypergraphs and computing fill-reducing orderings of sparse matrices. See URL `http://www-users.cs.umn.edu/~karypis/metis/`.

[13] P. S. Pacheco, *Parallel programming with MPI.* Morgan Kaufmann, San Francisco CA, 1997.

[14] S. Reitzinger and J. Schöberl, An algebraic multigrid method for finite element discretizations with edge elements. *Numer. Linear Algebra Appl.*, 9(3):223–238, 2002.

[15] M. Sala, M. A. Heroux, and D. D. Day, Trilinos 4.0 Tutorial. Technical Report SAND2004-2189, Sandia National Laboratories, May 2004.

[16] M. Sala, J. Hu, and R. S. Tuminaro, ML 3.1 Smoothed Aggregation User's Guide. Tech. Report SAND2004-4819, Sandia National Laboratories, September 2004.

[17] P. P. Silvester and R. L. Ferrari, *Finite Elements for Electrical Engineers.* Cambridge University Press, Cambridge, 3rd edition, 1996.

[18] G. L. G. Sleijpen and H. A. van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.

[19] The Trilinos Project Home Page, `http://software.sandia.gov/trilinos/`.

[20] P. Vaněk, J. Mandel, and M. Brezina, Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56(3):179–196, 1996.

[21] Zoltan Home Page, `http://www.cs.sandia.gov/Zoltan/`.