EFFICIENT CALCULATION OF EIGENMODES USING FEMAX++

R. Geus

PyFemax is a simulation tool being developed at PSI for computing eigenmodes of large RF structures. In 2003, additional functionality was added: the calculation of the Q-value, the implementation of first order edge elements and the integration of the LOBPCG eigensolver. Due to performance bottlenecks in the Python code of PyFemax, the application was rewritten from scratch in C++. The new code, Femax++, performs substantially better in large simulations, typically reducing simulation times by 50%. The parallel version of Femax++, whose development has started in December 2003, will enable PSI to run much larger simulations on a variety of parallel machines.

INTRODUCTION

PyFemax is a simulation tool for computing eigenmodes of large RF structures [3]. It is the result of an ongoing collaboration of PSI and the Institute of Computational Sciences at ETH Zürich. PyFemax uses unstructured tetrahedral grids and Nédélec finite elements to discretise Maxwell's equations. Several algorithms dealing with spurious modes are incorporated. The Jacobi-Davidson algorithm (JDSYM) is used for computing selected eigenpairs of the resulting large sparse symmetric eigenvalue problem. PyFemax offers some visualisation and postprocessing features.

PyFemax is implemented using the Pythonic approach [2], a combination of the interpreted Python programming language and the C programming language.

PyFemax has been successfully used to compute a few of the lowest eigenmodes of both the COMET cyclotron and the new PSI ring cyclotron cavity [4]. PyFemax is able to deal with large problems involving over eight million degrees of freedom.

At PSI PyFemax (and its successor Femax++) will be used complementarily to commercial tools like ANSYS, Microwave Studio and HFSS in cases where those tools cannot provide satisfactory results or when special customisation is required.

NEW FEATURES IN PYFEMAX

Recent experiments with box–shaped and cylindrical cavities (for which analytical solutions are known) have indicated that second order finite elements only yield the expected convergence behaviour if the tetrahedral mesh can represent the RF structure accurately. With the presence of curved boundaries however the favourable properties of second order elements are lost. To address this issue, first order Nédélec elements were implemented in PyFemax. These finite elements can provide a more economical alternative for complicated RF structures.

PyFemax's postprocessing features have been upgraded. A long requested feature for assessing the power loss in a cavity, the calculation of the quality factor, has been implemented. The main part of this computation is the evaluation of the surface integral

$$I = \iint_{\partial\Omega} \mathbf{curl}^2\, \mathbf{E}\; ds. \tag{1}$$

To this end we sum the analytically calculated integral contributions of all surface triangles of the tetrahedral mesh. We validated our implementation using the analytic box case and also by comparison with quality factors computed by ANSYS.

Finally, the LOBPCG eigensolver [6] was incorporated into PyFemax as an alternative to the Jacobi-Davidson algorithm. In our experiments [1] both solvers performed roughly the same. However, in a parallel context we expect LOBPCG to scale better than JDSYM.

THE NEW C++ IMPLEMENTATION: FEMAX++

When solving large problems involving several million DOFs, some performance bottlenecks in PyFemax became apparent. In particular, all mesh handling routines used in pre- and postprocessing were slow since the mesh data structures were implemented in Python. Instead of specifically optimising these critical sections in PyFemax, we decided to rewrite the program from scratch in C++ using an object–oriented programming style. This strategy offers several advantages:

- All Python related performance bottlenecks are removed.
- The Standard Template Library (STL), which is available for all modern C++ compilers, allows to port high–level Python data types like lists and dictionaries easily.
- The integration of Fortran 77, C and C++ libraries is straight–forward.
- C++ allows for a smoother transition to the planned parallel version than Python.

The code handling the tetrahedral mesh has been completely redesigned for Femax++. The old data structures which were implemented using several separate arrays holding e.g. coordinate data, mesh geometry and boundary condition information were replaced by a hierarchy of C++ classes. These classes represent points, edges, faces, tetrahedra and meshes.

The new data structure is much easier to extend, e.g. by new attributes or by new finite element types. The data structure now stores neighbour information (each tetrahedron knows its neighbour tetrahedra) which will be used to devise efficient algorithms for locating the tetrahedron associated with a given location (x, y, z). This functionality is fundamental for evaluating the computed eigenmodes at arbitrary locations in the cavity. Apart from these modifications, the program structure of PyFemax was only refined slightly. Major parts of the C and Python code could be refactored into the new C++ program in an efficient way. The rewriting process took less than 4 months.

As wrapper generation tools like SWIG, SIP, Boost.Python or Babel have dramatically improved over the last two years, it is now possible to generate a Python API layer for the new C++ program with only very little manual coding. We will use this technology to provide a convenient computational steering mechanism at a later stage.

	PyFemax		Femax++	
	t_{eig}	t_{proc}	t_{eig}	t_{proc}
cop10k	273.6	50.0	117.2	8.6
cop40k	2083.0	290.6	1115.8	70.8
box60k	2618.0	520.7	941.0	46.5
box170k	9064.6	2381.0	3343.1	114.4
cop300k	32598.1	6098.0	20977.5	513.6

Tab. 1: Simulation times for PyFemax and Femax

Tab. 1 shows simulation times (measured in seconds on the HP superdome *stardust* at ETH Zürich) for several discretisations of PSI ring cyclotron cavities using both PyFemax and Femax++. In this experiment the five lowest eigenmodes were computed to an accuracy of 10^{-6}. The solution times t_{eig} are 35%–65% smaller for Femax++. The improvement is even more dramatic for the pre- and postprocessing phase: the execution times t_{proc} are typically reduced by a factor of 10. For the largest grid with 300'000 second order finite elements the overall improvement is almost 50%.

PARALLELISATION

The parallelisation of Femax++ using the message–passing paradigm is the next major goal of this project. The development has started in December 2003. With the new parallel Femax++ code, much larger problems in the order of several tens of millions DOFs can be tackled. The parallel code will run on a wide variety of computers. The primary target architecture are workstation clusters, but the code will also support single processor mode and can run on large shared memory computers.

Femax++ uses the object-oriented software framework Trilinos, which is being developed at Sandia National Laboratories [5]. Trilinos provides parallel solver algorithms and libraries for the solution of large-scale, complex multi-physics engineering and scientific applications. Although Trilinos and the existing code in Femax++ have some overlapping functionality we expect a great reduction of development time from the use of Trilinos, since it provides distributed vector and matrix data structures and a large number of parallel solvers and preconditioners. Trilinos is still under active development and thus further improvements can be expected.

OUTLOOK AND CONCLUSIONS

The added functionality is an important step towards production use of PyFemax/Femax++. By rewriting the code in C++, we successfully removed the performance bottlenecks of PySparse, resulting in a overall runtime improvement of 50% for large simulations. The rewritten program is also easier to parallelise. We expect to deliver a first running version of parallel Femax++ with limited functionality in March 2004. Benchmarks will then be conducted to asses the scalability of our code and the Trilinos library on several parallel machines. Problematic program sections will then be optimised for the parallel environment. Once the prototype delivers satisfactory performance, the missing functionality (mainly postprocessing) will be merged to the parallel code. Later additional functionality, like e.g. the calculation of the gap voltage, will be implemented.

REFERENCES

[1] P. Arbenz and R. Geus. Multilevel preconditioners for solving eigenvalue problems occuring in the design of resonant cavities. Tech. Report 396, ETH Zürich, Institute of Scientific Computing, April 2003.

[2] O. Bröker, O. Chinelatto, and R. Geus. Using Python for large scale linear algebra applications. Submitted to Future Generation Computer Systems.

[3] R. Geus. *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities*. PhD thesis, Swiss Federal Institute of Technology Zurich, December 2002. Diss. ETH No. 14734.

[4] R. Geus, P. Arbenz, and L. Stingelin. PyFemax: A Python finite element Maxwell solver. PSI Scientific and Technical Report, 2002.

[5] M. Heroux, R. Bartlett, V. H. 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 Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[6] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. Tech. Report UCD-CCM 149, University of Colorado at Denver, Center for Computational Mathematics, 2000. (To appear in SIAM J. Sci. Comput.).