PARALLEL EIGENMODE COMPUTATIONS USING FEMAXX

R. Geus

We report on the progress of the development of the parallel eigenmode solver femaXX*. This project is a collaborative effort of PSI and the Institute of Computational Science at ETH Zurich. femaXX is the parallelised successor of pyfemax and femax++ [4]. It is intended to run very large scale eigenmode, quality factor and gap voltage computations for complicated RF structures like the one in the COMET cyclotron.*

PARALLEL IMPLEMENTATION

femaXX is a parallel code for execution on distributed memory architectures and uses the MPI (Message Passing Interface) standard for communication between the processes, which is supported by all modern parallel computers.

femaXX uses the same core computational algorithms as pyfemax and femax++ (Jacobi-Davidson algorithm and multilevel preconditioner). However the parallelisation for distributed memory machines made it necessary to rewrite femaXX from scratch: the data must be distributed explicitly over a series of processors and an efficient parallel implementation of the algorithms is necessary. Proper distributed data structures and data layout are required. Parallelised versions of numerical kernels such as direct and iterative solvers and preconditioners must be integrated. For our project, we found the Trilinos Project [5] to be a suitable environment to develop such a complex parallel application.

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, multi-physics engineering and scientific applications. 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 Trilinos packages and third party libraries

- Epetra, the fundamental Trilinos package for basic parallel algebraic operations. It provides a common infrastructure to the higher level packages,
- SuperLU_DIST, a parallel direct solver for sparse matrices and its Trilinos wrapper Amesos,
- AztecOO, an object-oriented descendant of the Aztec library of parallel iterative solvers and preconditioners,
- ML, the multilevel method package, that implements a smoothed aggregation AMG preconditioner capable of handling Maxwell equations,
- ParMetis, a package for graph partitioning used for distributing the sparse matrix data across the processors to minimise the communication volume and to balance the memory consumption,
- BLAS and LAPACK, basic sequential dense linear algebra routines.

GAP VOLTAGE CALCULATIONS

In order to calculate the gap voltage between point A and point B, the curve integral

$$\int_B^A \mathbf{E}(\mathbf{x})\mathbf{ds} \tag{1}$$

is evaluated along the particle trajectory. Since the trajectory can be arbitrary, the integral is computed numerically. The code must be able to evaluate the electric field $\mathbf{E}$ at any given location $\mathbf{x}$. Thus the tetrahedral finite element containing the point $\mathbf{x}$ must be identified. Since we are dealing with meshes containing several million tetrahedra it is crucial to have efficient algorithms and data structures for this task.

We chose to store all tetrahedra in a *loose octree*. An octree is a tree in which each node has 8 children. Each node of the tree corresponds to a cubical region, an octant. The root node represents a cube containing the whole mesh. Then, recursively, the eight children of each node represent the eight sub-cubes of the parent. In a loose octree, each octant child of the octree actually overlaps its siblings by a factor of 0.5. The tetrahedra are stored in leaf and node octants. Both the centre and extent of the bounding box of a tetrahedral element determine the actual octant storing it.

Given location $\mathbf{x}$, a short list of potential tetrahedra is obtained by recursively descending the octree and checking the bounding box of all tetrahedra in each visited octant. The final tetrahedron is found by properly testing all potential tetrahedra for containing point $\mathbf{x}$.

The loose octree data structure is very effective. For a mesh with 1.2 million tetrahedra, only 140 tetrahedra were visited on average to find the one containing $\mathbf{x}$. The list of potential tetrahedra contained 8 elements on average. On a 1.8 GHz AMD Opteron workstation one such lookup took 440 microseconds on average.

Since the function $\mathbf{E}(\mathbf{x})$ has low polynomial order, but is only piecewise contiguous, we chose the Adaptive Simpson Quadrature routine *adaptsim* by Gander and Gautschi [3] for computing the curve integral (1). We found adaptsim to be about three orders of magnitude faster than non-adaptive algorithms like Romberg or the trapezoidal rule.

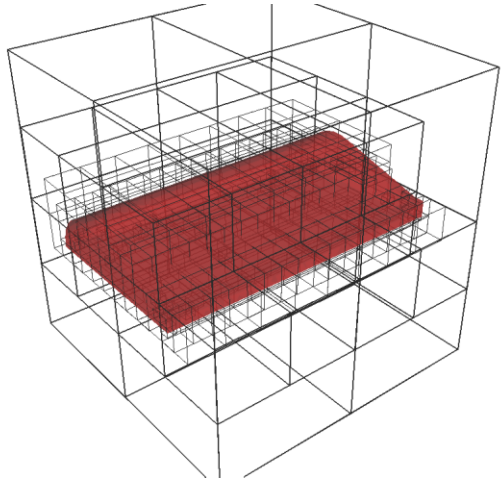

Fig. 1: Octree computed for a mesh of the copper cavity of the 590 MeV cyclotron with roughly 8'600 tetrahedra. The octree has 512 octants and a maximal depth of 5.

EXPERIMENTAL RESULTS

The following experiments have been conducted on *merlin*, the 32 dual node Linux workstation cluster at PSI. Each node has two AMD Athlon 1.4 GHz processors and 2 GB main memory. The nodes are connected by Myrinet, providing a bandwidth of 2 GBit/s.

To assess the parallel scalability of femaXX we compute the five lowest eigenmodes for the copper RF cavity of the 590 MeV cyclotron at PSI. A mesh with roughly 300'000 second order tetrahedral elements is used, resulting in 1.8 million degrees of freedom.

For this calculation the Jacobi-Davidson eigenvalue solver is used together with a combination of a hierarchical basis preconditioner and an algebraic multigrid (AMG) preconditioner. Both the eigenvalue solver and the preconditioner are customised to the Maxwell eigenvalue problem [2].

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

Table 1: Time spent in eigensolver, parallel efficiency and iteration counts for 8, 12 and 16 processors

The results in Table 1 show that, for these experiments, the iteration counts behave nicely and that efficiencies stay high.

As a second experiment we compute the lowest five eigenmodes of the RF structure of the COMET cyclotron with varying stem heights. The mesh consists of 1.2 million first order tetrahedral elements, resulting in 1.4 million degrees of freedom. We use the AMG preconditioner (ML) for the entire matrix in this case.

We start the experiment with stem heights of 219, 240, 240 and 235 millimetres and then change the heights with increments of 3 millimetres. The three lowest eigenfrequencies are reported in Table 2.

Δ_1	Δ_2	Δ_3	Δ_4	f_1	f_2	f_3
$+0$	$+0$	$+0$	$+0$	72.689	72.981	73.204
-3	$+0$	$+0$	$+0$	72.740	73.071	73.434
-6	$+0$	$+0$	$+0$	72.756	73.092	73.769
$+3$	$+0$	$+0$	$+0$	72.501	72.857	73.146
$+6$	$+0$	$+0$	$+0$	72.183	72.818	73.132
$+0$	$+0$	$+0$	-3	72.721	73.106	73.416
$+0$	$+0$	$+0$	-6	72.730	73.127	73.754
$+0$	$+0$	$+0$	$+3$	72.528	72.812	73.166
$+0$	$+0$	$+0$	$+6$	72.208	72.773	73.157
$+0$	-3	-3	$+0$	72.834	72.981	73.426
$+0$	-6	-6	$+0$	72.894	72.981	73.734
$+0$	$+3$	$+3$	$+0$	72.428	72.981	73.098
$+0$	$+6$	$+6$	$+0$	72.105	72.981	73.055

Table 2: The three lowest eigenfrequencies [MHz] computed for various adjustments of the four stem heights.

CONCLUSIONS AND OUTLOOK

The experimental results show that femaXX is capable of computing extremal eigenmodes of large problems efficiently in parallel. Currently femaXX uses more memory than necessary, which prevents us from solving our largest models on typical Linux clusters with limited amount of RAM per node. This problem is mainly caused by inefficient data structures for the sparse matrices in Epetra. The Trilinos developers are already working on a solution to this problem. Also our own code will be further optimised with respect to both memory consumption and runtime.

We are working on coupling femaXX with the particle tracking engine IPL [1]. femaXX will provide IPL with realistic E- and B-fields in cavities and improve the accuracy of the calculated particle trajectories.

ACKNOWLEDGEMENTS

I would like to thank Peter Arbenz, Martin Becka, Tiziano Mengotti (Institute of Computational Science, ETH Zürich) and Ulrich Hetmaniuk (Sandia National Labs) for their valuable contribution to this project.

REFERENCES

[1] A. Adelmann, R. Geus, M. Zenon, *Accurate Particle Tracking in RF Structures*, PSI Scientific and Technical Report 2004, VI.

[2] P. Arbenz, M. Becka, R. Geus, U. Hetmaniuk and T. Mengotti, *On a Parallel Multilevel Preconditioned Maxwell Eigensolver*, submitted to Parallel Computing.

[3] W. Gander and W. Gautschi, *Adaptive Quadrature - Revisited*, BIT Vol. 40, No. 1, March 2000.

[4] R. Geus, *Efficient Calculation of Eigenmodes using Femax++*, PSI Scientific and Technical Report 2003, VI.

[5] Trilinos Project, http://software.sandia.gov/trilinos/.