Towards a Parallel Maxwell Eigensolver

Peter Arbenz[1] Martin Bečka[1] Roman Geus[2]
Ulrich Hetmaniuk[3]

[1]Institute of Computational Science, ETH Zürich

[2]Paul-Scherrer-Institute, Villigen

[3]Sandia National Laboratory, Albuquerque

PDEMAMIP'04, Slanchev Bryag BG, Sept 7–10, 2004.

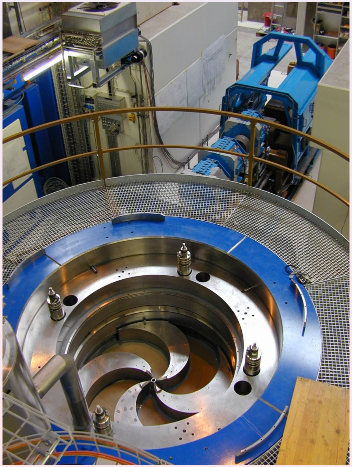
Statement of the problem

Maxwell equations (after separation of time/space variables and after elimination of the magnetic field intensity) become eigenvalue problem

$$\begin{aligned} \mathbf{curl}\,\mathbf{curl}\,\mathbf{e}(\mathbf{x}) &= \lambda\,\mathbf{e}(\mathbf{x}), & \mathbf{x} &\in \Omega, \\ \operatorname{div}\mathbf{e}(\mathbf{x}) &= 0, & \mathbf{x} &\in \Omega, \\ \mathbf{n}\times\mathbf{e} &= 0, & \mathbf{x} &\in \partial\Omega. \end{aligned} \tag{1}$$

Here, $\mathbf{e}$ is the electric field intensity.

View of a COMET cavity



COMET cyclotron for cancer therapy at PSI (3m diameter) produced by Accel Instruments GmbH (`www.accel.de`)

A mixed formulation

(Kikuchi 1987)

Find $(\lambda, \mathbf{e}, p) \in \mathbb{R} \times H_0(\mathbf{curl};\Omega) \times H_0^1(\Omega)$ *such that* $\mathbf{e} \neq \mathbf{0}$ *and*

$$\begin{array}{lll} \text{(a)} & (\mathbf{curl}\,\mathbf{e}, \mathbf{curl}\,\boldsymbol{\Psi}) + (\mathbf{grad}\,p, \boldsymbol{\Psi}) = \lambda(\mathbf{e}, \boldsymbol{\Psi}), & \forall \boldsymbol{\Psi} \in H_0(\mathbf{curl};\Omega) \\ \text{(b)} & (\mathbf{e}, \mathbf{grad}\,q) = 0, & \forall q \in H_0^1(\Omega) \end{array} \tag{2}$$

Here, p is a Lagrange multiplier.
(b) reflects the Helmholtz decomposition
$H_0(\mathbf{curl};\Omega) = W_0 \oplus \mathbf{grad}\,H_0^1(\Omega)$ where W_0 is the subset of divergence-free fields in $H_0(\mathbf{curl};\Omega)$.

The matrix eigenvalue problem

The evp for the time-harmonic Maxwell equation is given by

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad C^T\mathbf{x} = \mathbf{0}. \tag{3}$$

where $A = A^T \geq 0$ with huge nullspace, $M = M^T > 0$.
The elements of A, M, and C are

$$A_{ij} = (\mathbf{curl}\,\mathbf{\Psi}_i, \mathbf{curl}\,\mathbf{\Psi}_j), \quad M_{ij} = (\mathbf{\Psi}_i, \mathbf{\Psi}_j), \qquad 1 \leq i, j \leq n,$$

and

$$C_{ik} = (\mathbf{e}_i, \mathbf{grad}\, q_k), \quad 1 \leq i \leq n,\ 1 \leq k \leq m \approx n/6.$$

Here, the $\mathbf{\Psi}_j$ are Nédélec (edge) element basis functions (Nédélec,1980) and the q_k are Lagrange (node) finite elements. Both quadratic elements, with hierarchical bases.

Null space of A

A sparse basis Y of the nullspace of A can easily be given (incidence matrix). Then, $C = MY$.
The M-orthogonal projector onto $\mathcal{R}(A) = \mathcal{N}(A)^{\perp_M} = \mathcal{N}(C^T)$ is given by

$$I - YH^{-1}C^T \tag{4}$$

where H with elements

$$H_{kl} = (\mathbf{grad}\,\varphi_k, \mathbf{grad}\,\varphi_l)$$

is Poisson matrix for quadratic Lagrange elements on given FE mesh.
We execute our computations in $\mathcal{N}(C^T)$ to avoid computation of zero eigenvalues of (3) by applying (4) at the right places.

Symmetric Jacobi–Davidson (JDSYM)

(Sleijpen/van der Vorst, 1996; Geus, 2003)

- Let $\mathcal{V}_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathcal{N}(C^T)$, $\mathbf{v}_k^T M \mathbf{v}_j = \delta_{kj}$, be the actual search space (**not** a Krylov space).
- **Rayleigh–Ritz–Galerkin procedure:** Extract Ritz pair $(\tilde{\lambda}, \tilde{\mathbf{q}})$ in $\mathcal{V}_k$ with $\tilde{\lambda}$ 'closest' to some target value τ.
- **Convergence:** If $\|\mathbf{r}_k\|_{M^{-1}} \equiv \|(A - \tilde{\lambda} M)\,\tilde{\mathbf{q}}\|_{M^{-1}} < \varepsilon \|\tilde{\mathbf{q}}\|_M$ then we have found an eigenpair
- Solve **correction equation** for $\mathbf{t}_k \perp_M \tilde{\mathbf{q}}$,

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \quad \tilde{\mathbf{q}}^T M \mathbf{t}_k = 0. \quad (5)$$

- M-orthonormalize $(I - YH^{-1}C^T)\mathbf{t}_k$ to $\mathcal{V}_k$ to obtain $\mathbf{v}_{k+1}$
- Expand search space: $\mathcal{V}_{k+1} = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$.

Symmetric Jacobi–Davidson (JDSYM)

(Sleijpen/van der Vorst, 1996; Geus, 2003)

- Let $\mathcal{V}_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathcal{N}(C^T)$, $\mathbf{v}_k^T M \mathbf{v}_j = \delta_{kj}$, be the actual search space (**not** a Krylov space).
- **Rayleigh–Ritz–Galerkin procedure:** Extract Ritz pair $(\tilde{\lambda}, \tilde{\mathbf{q}})$ in $\mathcal{V}_k$ with $\tilde{\lambda}$ 'closest' to some target value τ.
- **Convergence:** If $\|\mathbf{r}_k\|_{M^{-1}} \equiv \|(A - \tilde{\lambda} M)\,\tilde{\mathbf{q}}\|_{M^{-1}} < \varepsilon \|\tilde{\mathbf{q}}\|_M$ then we have found an eigenpair
- Solve **correction equation** for $\mathbf{t}_k \perp_M \tilde{\mathbf{q}}$,

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \quad \tilde{\mathbf{q}}^T M \mathbf{t}_k = 0. \quad (5)$$

- M-orthonormalize $(I - YH^{-1}C^T)\mathbf{t}_k$ to $\mathcal{V}_k$ to obtain $\mathbf{v}_{k+1}$
- Expand search space: $\mathcal{V}_{k+1} = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$.

Symmetric Jacobi–Davidson (JDSYM)

(Sleijpen/van der Vorst, 1996; Geus, 2003)

- Let $\mathcal{V}_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathcal{N}(C^T)$, $\mathbf{v}_k^T M \mathbf{v}_j = \delta_{kj}$, be the actual search space (**not** a Krylov space).
- **Rayleigh–Ritz–Galerkin procedure:** Extract Ritz pair $(\tilde{\lambda}, \tilde{\mathbf{q}})$ in $\mathcal{V}_k$ with $\tilde{\lambda}$ 'closest' to some target value τ.
- **Convergence:** If $\|\mathbf{r}_k\|_{M^{-1}} \equiv \|(A - \tilde{\lambda} M)\,\tilde{\mathbf{q}}\|_{M^{-1}} < \varepsilon \|\tilde{\mathbf{q}}\|_M$ then we have found an eigenpair
- Solve **correction equation** for $\mathbf{t}_k \perp_M \tilde{\mathbf{q}}$,

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \quad \tilde{\mathbf{q}}^T M \mathbf{t}_k = 0. \quad (5)$$

- M-orthonormalize $(I - YH^{-1}C^T)\mathbf{t}_k$ to $\mathcal{V}_k$ to obtain $\mathbf{v}_{k+1}$
- Expand search space: $\mathcal{V}_{k+1} = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$.

Symmetric Jacobi–Davidson (JDSYM)

(Sleijpen/van der Vorst, 1996; Geus, 2003)

- Let $\mathcal{V}_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathcal{N}(C^T)$, $\mathbf{v}_k^T M \mathbf{v}_j = \delta_{kj}$, be the actual search space (**not** a Krylov space).
- **Rayleigh–Ritz–Galerkin procedure:** Extract Ritz pair $(\tilde{\lambda}, \tilde{\mathbf{q}})$ in $\mathcal{V}_k$ with $\tilde{\lambda}$ 'closest' to some target value τ.
- **Convergence:** If $\|\mathbf{r}_k\|_{M^{-1}} \equiv \|(A - \tilde{\lambda}M)\,\tilde{\mathbf{q}}\|_{M^{-1}} < \varepsilon \|\tilde{\mathbf{q}}\|_M$ then we have found an eigenpair
- Solve **correction equation** for $\mathbf{t}_k \perp_M \tilde{\mathbf{q}}$,

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \quad \tilde{\mathbf{q}}^T M \mathbf{t}_k = 0. \quad (5)$$

- M-orthonormalize $(I - YH^{-1}C^T)\mathbf{t}_k$ to $\mathcal{V}_k$ to obtain $\mathbf{v}_{k+1}$
- Expand search space: $\mathcal{V}_{k+1} = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$.

Symmetric Jacobi–Davidson (JDSYM)

(Sleijpen/van der Vorst, 1996; Geus, 2003)

- Let $\mathcal{V}_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathcal{N}(C^T)$, $\mathbf{v}_k^T M \mathbf{v}_j = \delta_{kj}$, be the actual search space (**not** a Krylov space).
- **Rayleigh–Ritz–Galerkin procedure:** Extract Ritz pair $(\tilde{\lambda}, \tilde{\mathbf{q}})$ in $\mathcal{V}_k$ with $\tilde{\lambda}$ 'closest' to some target value τ.
- **Convergence:** If $\|\mathbf{r}_k\|_{M^{-1}} \equiv \|(A - \tilde{\lambda} M)\,\tilde{\mathbf{q}}\|_{M^{-1}} < \varepsilon \|\tilde{\mathbf{q}}\|_M$ then we have found an eigenpair
- Solve **correction equation** for $\mathbf{t}_k \perp_M \tilde{\mathbf{q}}$,

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \quad \tilde{\mathbf{q}}^T M \mathbf{t}_k = 0. \quad (5)$$

- M-orthonormalize $(I - YH^{-1}C^T)\mathbf{t}_k$ to $\mathcal{V}_k$ to obtain $\mathbf{v}_{k+1}$
- Expand search space: $\mathcal{V}_{k+1} = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$.

Symmetric Jacobi–Davidson (JDSYM)

(Sleijpen/van der Vorst, 1996; Geus, 2003)

- Let $\mathcal{V}_k = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathcal{N}(C^T)$, $\mathbf{v}_k^T M \mathbf{v}_j = \delta_{kj}$, be the actual search space (**not** a Krylov space).
- **Rayleigh–Ritz–Galerkin procedure:** Extract Ritz pair $(\tilde{\lambda}, \tilde{\mathbf{q}})$ in $\mathcal{V}_k$ with $\tilde{\lambda}$ 'closest' to some target value τ.
- **Convergence:** If $\|\mathbf{r}_k\|_{M^{-1}} \equiv \|(A - \tilde{\lambda} M)\, \tilde{\mathbf{q}}\|_{M^{-1}} < \varepsilon \|\tilde{\mathbf{q}}\|_M$ then we have found an eigenpair
- Solve **correction equation** for $\mathbf{t}_k \perp_M \tilde{\mathbf{q}}$,

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \quad \tilde{\mathbf{q}}^T M \mathbf{t}_k = 0. \quad (5)$$

- M-orthonormalize $(I - YH^{-1}C^T)\mathbf{t}_k$ to $\mathcal{V}_k$ to obtain $\mathbf{v}_{k+1}$
- Expand search space: $\mathcal{V}_{k+1} = \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\}$.

Remarks on JDSYM

- Shift η_k is set to target value τ initially and to the Rayleigh quotient $\rho(\tilde{\mathbf{q}})$ close to convergence.
- If $k = j_{\max}$ reduce size of the search space to $j_{\min}$. Use $j_{\min}$ 'best' Ritz vectors in $\mathcal{V}_{j_{\max}}$ to define $\mathcal{V}_{j_{\min}}$.
- The correction equation is solved only approximatively. We use a Krylov space method: QMRS (admits indefinite preconditioner).
- Eigenvectors corresponding to higher eigenvalues are computed in the orthogonal complement of previously computed eigenvectors.

Remarks on JDSYM

- Shift η_k is set to target value τ initially and to the Rayleigh quotient $\rho(\tilde{\mathbf{q}})$ close to convergence.
- If $k = j_{\max}$ reduce size of the search space to $j_{\min}$. Use $j_{\min}$ 'best' Ritz vectors in $\mathcal{V}_{j_{\max}}$ to define $\mathcal{V}_{j_{\min}}$.
- The correction equation is solved only approximatively. We use a Krylov space method: QMRS (admits indefinite preconditioner).
- Eigenvectors corresponding to higher eigenvalues are computed in the orthogonal complement of previously computed eigenvectors.

Remarks on JDSYM

- Shift η_k is set to target value τ initially and to the Rayleigh quotient $\rho(\tilde{\mathbf{q}})$ close to convergence.
- If $k = j_{\max}$ reduce size of the search space to $j_{\min}$. Use $j_{\min}$ 'best' Ritz vectors in $\mathcal{V}_{j_{\max}}$ to define $\mathcal{V}_{j_{\min}}$.
- The correction equation is solved only approximatively. We use a Krylov space method: QMRS (admits indefinite preconditioner).
- Eigenvectors corresponding to higher eigenvalues are computed in the orthogonal complement of previously computed eigenvectors.

Remarks on JDSYM

- Shift η_k is set to target value τ initially and to the Rayleigh quotient $\rho(\tilde{\mathbf{q}})$ close to convergence.
- If $k = j_{\max}$ reduce size of the search space to $j_{\min}$. Use $j_{\min}$ 'best' Ritz vectors in $\mathcal{V}_{j_{\max}}$ to define $\mathcal{V}_{j_{\min}}$.
- The correction equation is solved only approximatively. We use a Krylov space method: QMRS (admits indefinite preconditioner).
- Eigenvectors corresponding to higher eigenvalues are computed in the orthogonal complement of previously computed eigenvectors.

Preconditioning the correction equation

The correction equation is given by

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)(A - \eta_k M)(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{t}_k = -\mathbf{r}_k, \qquad \tilde{\mathbf{q}}^T M\mathbf{t}_k = 0.$$

We choose a preconditioner of the form

$$(I - M\tilde{\mathbf{q}}\tilde{\mathbf{q}}^T)K(I - \tilde{\mathbf{q}}\tilde{\mathbf{q}}^T M)\mathbf{c} = \mathbf{b}, \qquad \tilde{\mathbf{q}}^T M\mathbf{c} = 0. \tag{6}$$

where K is a preconditioner for $A - \rho_k M$.
As we are looking for just a few of the smallest eigenvalues we take $K \approx A - \sigma M$ where σ is close to the desired eigenvalues.
We use the same K for all correction equations.

Hierarchical basis preconditioning

For solving with K we employ the hierarchical basis (Bank, 1996):
We arrange the matrix K in the form

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \qquad K = A - \sigma M. \tag{7}$$

In a 2-level algorithm K_{11} corresponds to a system on the coarser grid whence it is solved directly.
Here, solving with K_{11} is replaced by invoking the AMG multilevel solver ML in the Trilinos solver suite (Reitzinger/Schöberl, 2002; P. Vaněk et al., 2001; Tuminaro et al., 2004).

Solving with K is replaced by one step of symmetric block Gauss–Seidel iteration

$$\begin{aligned} \mathbf{x}_1' &:= K_{11}^{-1}\mathbf{b}_1, \\ \mathbf{x}_2 &:= (\widetilde{K}_{22})^{-1}(\mathbf{b}_2 - K_{21}\mathbf{x}_1'), \\ \mathbf{x}_1 &:= K_{11}^{-1}(\mathbf{b}_1 - K_{12}\mathbf{x}_2), \end{aligned} \tag{8}$$

with $\widetilde{K}_{22}$ only an approximation of K_{22}, $\widetilde{K}_{22} \approx K_{22}$. Here, $\widetilde{K}_{22}$ corresponds to one step of (undamped) Jacobi iteration.

The Software Environment: Trilinos

- The Trilinos Project is an effort to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications.
- See `http://software.sandia.gov/trilinos/`
- Provides means to distribute (multi)vectors and (sparse) matrices (Epetra and EpetraExt packages).
- Provides solvers that work on these distributed data. Here we use iterative solvers and preconditioners (package AztecOO), smoothed aggregation multilevel AMG preconditioner (ML), direct solver wrappers (Amesos) and data distribution for parallelization (EpetraExt interface to Zoltan/ParMETIS).
- Quality of documentation depends on package.

The Software Environment: Trilinos

- The Trilinos Project is an effort to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications.
- See `http://software.sandia.gov/trilinos/`
- Provides means to distribute (multi)vectors and (sparse) matrices (Epetra and EpetraExt packages).
- Provides solvers that work on these distributed data. Here we use iterative solvers and preconditioners (package AztecOO), smoothed aggregation multilevel AMG preconditioner (ML), direct solver wrappers (Amesos) and data distribution for parallelization (EpetraExt interface to Zoltan/ParMETIS).
- Quality of documentation depends on package.

The Software Environment: Trilinos

- The Trilinos Project is an effort to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications.
- See `http://software.sandia.gov/trilinos/`
- Provides means to distribute (multi)vectors and (sparse) matrices (Epetra and EpetraExt packages).
- Provides solvers that work on these distributed data. Here we use iterative solvers and preconditioners (package AztecOO), smoothed aggregation multilevel AMG preconditioner (ML), direct solver wrappers (Amesos) and data distribution for parallelization (EpetraExt interface to Zoltan/ParMETIS).
- Quality of documentation depends on package.

The Software Environment: Trilinos

- The Trilinos Project is an effort to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications.
- See `http://software.sandia.gov/trilinos/`
- Provides means to distribute (multi)vectors and (sparse) matrices (Epetra and EpetraExt packages).
- Provides solvers that work on these distributed data. Here we use iterative solvers and preconditioners (package AztecOO), smoothed aggregation multilevel AMG preconditioner (ML), direct solver wrappers (Amesos) and data distribution for parallelization (EpetraExt interface to Zoltan/ParMETIS).
- Quality of documentation depends on package.

The Software Environment: Trilinos

- The Trilinos Project is an effort to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications.
- See `http://software.sandia.gov/trilinos/`
- Provides means to distribute (multi)vectors and (sparse) matrices (Epetra and EpetraExt packages).
- Provides solvers that work on these distributed data. Here we use iterative solvers and preconditioners (package AztecOO), smoothed aggregation multilevel AMG preconditioner (ML), direct solver wrappers (Amesos) and data distribution for parallelization (EpetraExt interface to Zoltan/ParMETIS).
- Quality of documentation depends on package.

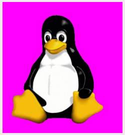
Example of using Trilinos

```
// Example of solving a linear system with AztecOO.

// create a linear map
Epetra_Map RowMap(NumGlobalElements, 0,
Communicator);

// create an Epetra_Matrix
Epetra_CrsMatrix A(Copy, RowMap, NumEntriesPerRow);

// fill a row with values
A.InsertGlobalValues(GlobalRow, NumEntries, Values,
Indices);
```

```
// ========== AZTECOO INTERFACE ============
// create vectors x and b
Epetra_Vector x(Map);
Epetra_Vector b(Map);
b.Random();

// create a linear problem
Epetra_LinearProblem Problem(&A, &x, &b);

// create an AztecOO instance
AztecOO Solver(Problem);

Solver.SetAztecOption(AZ_precond, AZ_Jacobi);
Solver.Iterate(1000, 1E-9);
```

BeoWulf Cluster

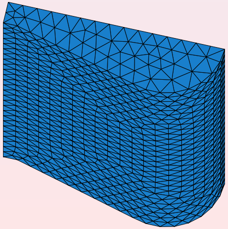
- 32 dual-node PC cluster
  - 2 AMD Athlon 1.4 GHz processors/node
  - 2 GB main memory
  - 160 GB local disk
- Myrinet
  - 2000 Mbit/s
- Software
  - Linux 2.4.20
  - MPICH 1.2.5
  - Trilinos Developer Version (March)

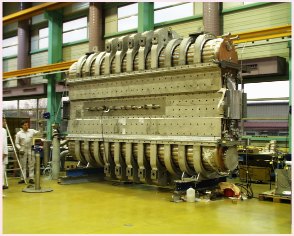


Matrices

grid	$n_{A-\sigma M}$	$nnz_{A-\sigma M}$	n_H	nnz_H
cop40k	231668	4811786	46288	1163834
box170k	1030518	20767052	209741	5447883

Table: Matrices used for numerical experiments





Data distribution

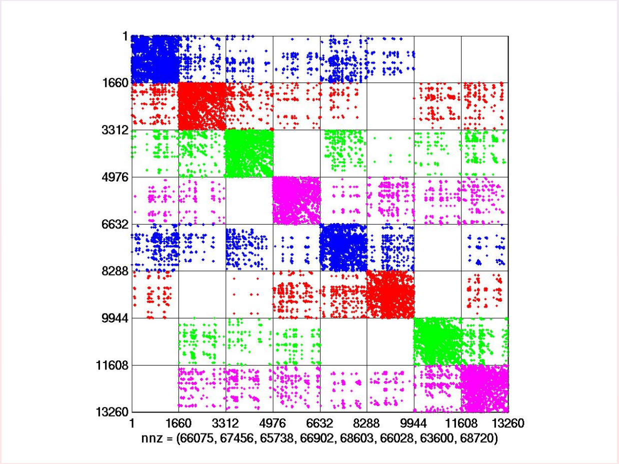
- **Definition of an artificial graph**
  The blocks of A, M, K, H, and C are stored *independently*. Distribution is by rows. A map defines which row goes on which processor.
  For the distribution with ParMETIS a 'graph' G is defined that contains a node for each vertex, edge, and face of the finite element mesh that 'participates' at the computation. G is defined by suitable submatrices of M, H, and C.

More precisely we define a graph G as

$$G = \begin{bmatrix} H_{11} & C_{11}^T & \widehat{C}_{21}^T \\ C_{11} & M_{11} & \widehat{M}_{12} \\ \widehat{C}_{21} & \widehat{M}_{21} & \widehat{M}_{22} \end{bmatrix}. \tag{9}$$

ParMETIS tries to distribute the matrix such that the number of nonzero elements per processor is balanced (load balance) and such that the number of edge cuts is minimal (little communication).

- **Typical distribution of M on 8 processors**



matrix order 13'260
# nonzeros 533'122
533'122/8 = 66'640
85-90% of nonzeros
in block diagonal

Timings

Timings are given for computing the 5 smallest positive eigenvalues using JDSYM with the 2-level preconditioner (K_{11}: ML, K_{22}: diagonal) on the Beowulf (Merlin) in dedicated mode. System with H was solved using PCG with ML preconditioner.

Some JDSYM parameters

```
itmax=200 linitmax=50 kmax=5 jmin=6 jmax=15
tau=0.0e+00 jdtol=1.0e-08 eps_tr=1.0e-03
toldecay=1.5e+00 sigma=1.5e+00
linsolver=qmrs blksize=1
```

	cop40k					
p	t [sec]	$E(p)$[1]	t(Prec)	t(Proj)	n_{outer}	$n_{\text{inner}}^{\text{avg}}$
2	1241	1.00	38%	16%	55	19.38
4	637	0.97	37%	17%	54	19.24
6	458	0.90	39%	18%	54	19.69
8	330	0.94	39%	17%	53	19.53
10	266	0.93	39%	19%	52	19.17
12	240	0.86	41%	20%	54	19.61
14	211	0.84	42%	20%	55	19.36
16	186	0.83	44%	20%	54	19.17

[1] Efficiency relative to execution time $t(2)$

	box170k					
p	t [sec]	$E(p)$[2]	t(Prec)	t(Proj)	n_{outer}	$n_{\text{inner}}^{\text{avg}}$
2	—	—	—	—	—	—
4	7720	1.00	28%	22%	54	22.39
6	2237	2.30	39%	23%	55	22.47
8	1744	2.21	38%	23%	55	23.51
10	1505	2.05	38%	25%	56	22.54
12	1224	2.10	38%	25%	54	22.02
14	1118	1.97	39%	24%	55	23.76
16	932	2.07	38%	25%	54	22.30

[2] Efficiency relative to execution time $t(4)$

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximately with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.

Summary

1. We presented some preliminary results on a parallel implementation of the symmetric Jacobi-Davidson algorithm (JDSYM).
2. Ingredients of the present parallel JDSYM are
   - Trilinos framework
   - Data distribution by Zoltan/ParMETIS
   - Correction equations are solved approximatively with QMRS and a 2-level hierarchical basis preconditioner enriched with the aggregated multilevel preconditioner ML
   - Equations on coarsest grid are solved with SuperLU
3. We should improve the K_{22} solver and the matrix (re)distribution.
4. Tuning the code and more extensive experiments are required.