Implementation of a Geometric Multigrid Method for FEniCS and its Application

Felix Ospald

Chemnitz University of Technology, Faculty of Mathematics
Research Group Numerical Mathematics (Partial Differential Equations)

FEniCS‘13
Cambridge, UK
March 18, 2013

Outline

http://launchpad.net/fmg

Introduction to Geometric Multigrid (GMG)

Overview

Multigrid methods are efficient solvers/preconditioners for linear or nonlinear systems of equations coming from a discretization of a (elliptic) PDE, i.e. find $u \in U$ such that

$$a(u, v) = f(v) \quad \forall v \in V,$$

or equivalently

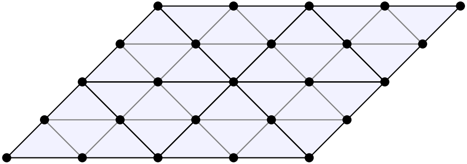
$$Ax = b.$$

They have a numerical complexity of $\mathcal{O}(N)$, if used in the right way.

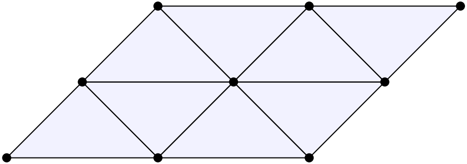
(U: trial space, V: test space)

Multigrid Idea

Geometric multigrid is based on a fine-to-coarse grid/FE-space hierarchy of the problem.



U_1, V_1
matrix: A_1, rhs: b



$U_0 \subset U_1, V_0 \subset V_1$
matrix: A_0

Multigrid Idea

Let x be some initial guess of the solution $x^* = A_1^{-1} b$.

1. Reduce the high frequency components of the error $e = x^* - x$ and residual $r = b - A_1 x$ by smoothing $x := \mathcal{S}_1^{\nu_1}(x, b)$.
2. Now the error/residual can be well approximated on the coarse grid as $r_0 = Rr$, by using the restriction operator $R : V_1' \mapsto V_0'$.
3. Solve the residual equation $A_0 e = r_0$ (on the coarse grid).
4. Interpolate the error e onto the fine grid, by using the prolongation operator $P : U_0 \mapsto U_1$.
5. Improve the current solution $x := x + Pe$
6. Smooth again $x := \mathcal{S}_2^{\nu_2}(x, b)$

Note: Residual Equation

$$r = b - A_1 x = (b - A_1 x) - (b - A_1 x^*) = A_1(x^* - x) = A_1 e.$$

The Two-Grid Method

This gives the two-grid method:

Algorithm 1: The two-grid method $TGM(x, b)$.

```
1 x := S_1^{ν1}(x, b)                                      // pre-smoothing
2 r_0 := R(b − A_1 x)      // residual computation + restriction
3 e := A_0^{−1} r_0                               // solve coarse problem
4 x := x + Pe                           // prolongation + correction step
5 x := S_2^{ν2}(x, b)                                     // post-smoothing
6 return x
```

1. $x := \mathcal{S}_1^{\nu_1}(x, b)$ — // pre-smoothing
2. $r_0 := R(b - A_1 x)$ — // residual computation + restriction
3. $e := A_0^{-1} r_0$ — // solve coarse problem
4. $x := x + Pe$ — // prolongation + correction step
5. $x := \mathcal{S}_2^{\nu_2}(x, b)$ — // post-smoothing
6. **return** x

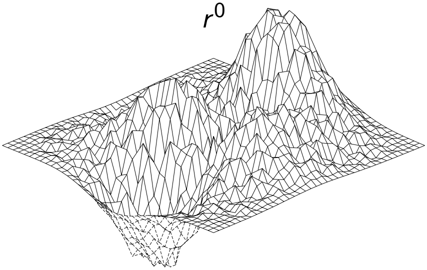
Applying the TGM recursively (solving the equation in line 3) gives the multi-grid method.

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$x^{k+1} = x^k + M^{-1} r^k,$$
$$r^{k+1} = (I - AM^{-1}) r^k,$$
$$e^{k+1} = (I - M^{-1}A) e^k,$$



where

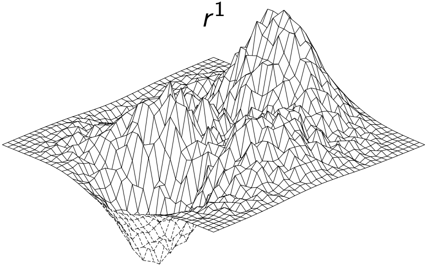
$M = \omega^{-1} I,$	for relaxed Richardson,
$M = \omega^{-1} D,$	for relaxed Jacobi,
$M = \omega^{-1} D + L,$	for relaxed Gauß-Seidel (SOR).

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$x^{k+1} = x^k + M^{-1} r^k,$$
$$r^{k+1} = (I - AM^{-1}) r^k,$$
$$e^{k+1} = (I - M^{-1}A) e^k,$$



where

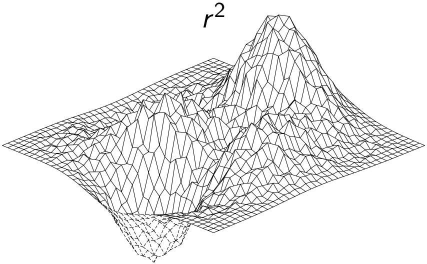
$M = \omega^{-1} I,$	for relaxed Richardson,
$M = \omega^{-1} D,$	for relaxed Jacobi,
$M = \omega^{-1} D + L,$	for relaxed Gauß-Seidel (SOR).

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$x^{k+1} = x^k + M^{-1} r^k,$$
$$r^{k+1} = (I - AM^{-1}) r^k,$$
$$e^{k+1} = (I - M^{-1} A) e^k,$$



where

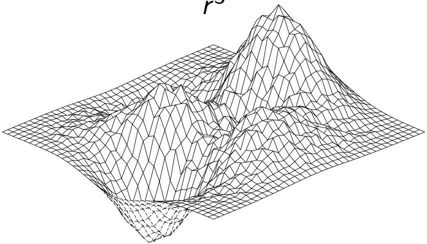
$M = \omega^{-1} I,$	for relaxed Richardson,
$M = \omega^{-1} D,$	for relaxed Jacobi,
$M = \omega^{-1} D + L,$	for relaxed Gauß-Seidel (SOR).

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$x^{k+1} = x^k + M^{-1} r^k,$$
$$r^{k+1} = (I - AM^{-1}) r^k,$$
$$e^{k+1} = (I - M^{-1}A) e^k,$$



where

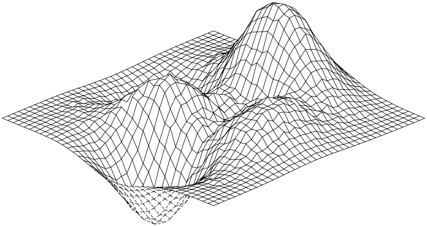
$M = \omega^{-1} I,$	for relaxed Richardson,
$M = \omega^{-1} D,$	for relaxed Jacobi,
$M = \omega^{-1} D + L,$	for relaxed Gauß-Seidel (SOR).

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$x^{k+1} = x^k + M^{-1} r^k,$$
$$r^{k+1} = (I - AM^{-1}) r^k,$$
$$e^{k+1} = (I - M^{-1}A) e^k,$$



where

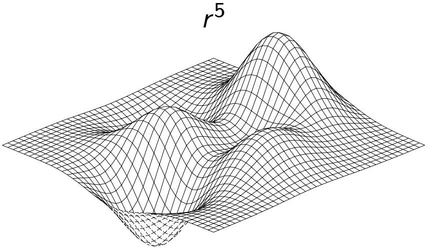
$M = \omega^{-1} I,$	for relaxed Richardson,
$M = \omega^{-1} D,$	for relaxed Jacobi,
$M = \omega^{-1} D + L,$	for relaxed Gauß-Seidel (SOR).

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$
\begin{aligned}
x^{k+1} &= x^k + M^{-1} r^k, \\
r^{k+1} &= (I - AM^{-1}) r^k, \\
e^{k+1} &= (I - M^{-1}A) e^k,
\end{aligned}
$$



where

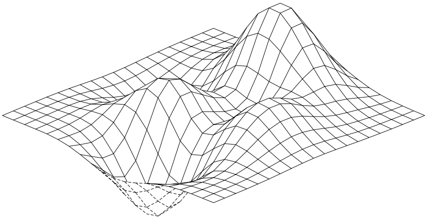
$$
\begin{aligned}
M &= \omega^{-1} I, && \text{for relaxed Richardson,} \\
M &= \omega^{-1} D, && \text{for relaxed Jacobi,} \\
M &= \omega^{-1} D + L, && \text{for relaxed Gauß-Seidel (SOR).}
\end{aligned}
$$

Smoothing

Reduces the high frequency components of the error/residual.

Basic smoothers are of the form

$$x^{k+1} = x^k + M^{-1} r^k,$$
$$r^{k+1} = (I - AM^{-1}) r^k,$$
$$e^{k+1} = (I - M^{-1}A) e^k,$$

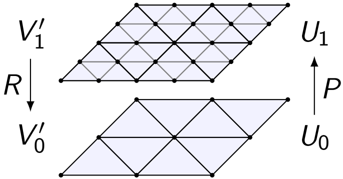


where

$M = \omega^{-1} I,$	for relaxed Richardson,
$M = \omega^{-1} D,$	for relaxed Jacobi,
$M = \omega^{-1} D + L,$	for relaxed Gauß-Seidel (SOR).

Restriction and Prolongation

Transfer functions (solutions) and functionals (right hand sides) between two grids.



Prolongation: Interpolate a function from U_0 to U_1. Since $U_0 \subset U_1$ it is obvious to use the injection $i_U : U_0 \hookrightarrow U_1$ for P.

Restriction: Restrict a functional from V_1' to V_0'. Since $V_0 \subset V_1$ it seems to be natural to use the restriction $\circ i_V$ for R (i.e. $Rr = r \circ i_V$ with the injection $i_V : V_0 \hookrightarrow V_1$).

Multigrid and FEniCS

FEniCS already comes with some algebraic multigrid (AMG) preconditioners via PETSc (Hypre, Sandia ML).

Difference between AMG and GMG

AMG preconditioners only get the matrix A_1 and derive some A_0, P, R, etc. from this matrix (black box solver). In contrast to GMG, AMG does not use information about the FE-spaces, which is actually available in FEniCS.

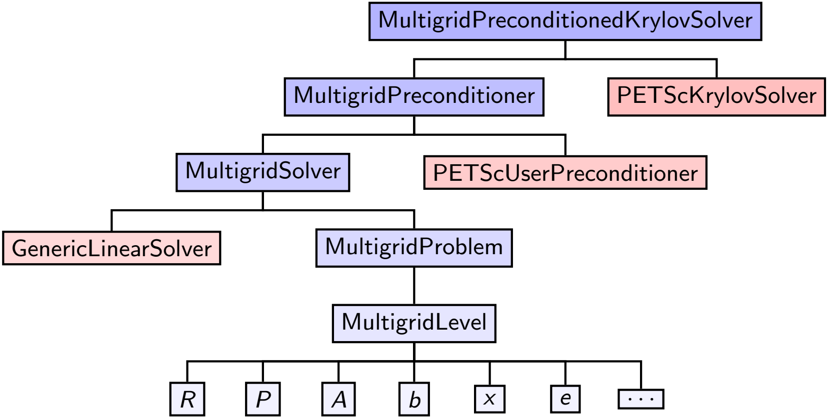
Since it is very easy to construct problem hierarchies in FEniCS, it is very reasonable to use GMG methods in FEniCS.

Implementation

Features

- language: C++
- easily useable and extendable
- relatively fast
- works for all nested FE-families in FEniCS (excludes Crouzeix-Raviart)
- can be used as iterative multigrid solver and preconditioner for Krylov-subspace methods (CG, MINRES, ...)
- supports uniform and local mesh refinement

Classes


MultigridPreconditionedKrylovSolver
MultigridPreconditioner
PETScKrylovSolver
MultigridSolver
PETScUserPreconditioner
GenericLinearSolver
MultigridProblem
MultigridLevel
R
P
A
b
x
e
· · ·

Code Examples (C++)

How to do it with DOLFIN

```
...
LinearVariationalProblem problem(a, L, u, bc);
LinearVariationalSolver solver(problem);
solver.solve();
plot(problem.solution());
...
```

Using FMG

```
#include <fmg.h>
...
LinearVariationalProblem problem(a, L, u, bc);
fmg::MultigridPreconditionedKrylovSolver solver(problem, 4);
solver.solve();
plot(solver.solution());
...
```

Code Examples (C++)

Advanced Example

```
...
LinearVariationalProblem problem(a, L, u, bc);
fmg::MultigridProblem mg_problem(problem);
mg_problem.adapt();
mg_problem.adapt();
...

fmg::MultigridSolver mg_solver(mg_problem);
mg_solver.parameters["pre_smoother"] = "jacobi";
mg_solver.parameters["pre_smoother_relax"] = 0.6;
mg_solver.parameters["post_smoother"] = "jacobi";
mg_solver.parameters["post_smoother_relax"] = 0.6;
mg_solver.parameters["coarse_solver_type"] = "lu";
solver.solve();
...
```

Code Examples (C++)

Performance Testing

```
...
LinearVariationalProblem problem(a, L, u, bc);

fmg::Tests tests(problem);

tests.parameters["test_solver"] = "cg+fmg,cg+hypre_amg";
tests.parameters["test_smoothers"] = "jacobi@0.6,fsor+bsor";
tests.parameters["num_refinements"] = 4;
tests.parameters.parse(argc, argv);

tests.run();
```

Command line call:

```
./main --num_refinements 5 --table_format latex
```

Numerical Results

Numerical Results

Comparison of

- FMG GMG,
- Hypre AMG (PETSc preconditioner),
- ML AMG (PETSc preconditioner),

as a preconditioner for CG/MINRES in terms of

- setup time (initialization of the coarse grid problems and grid transfer operators),
- solve time,
- number of iterations

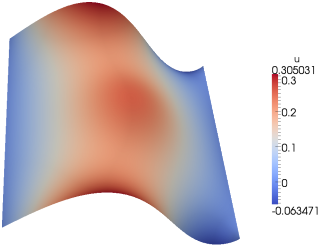
Numerical Results

1. Poisson problem (2D)
2. linear Elasticity (3D)
3. Stokes problem (2D)

Poisson Problem (2D)

Poisson problem with mixed boundary conditions (FEniCS demo):

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \Gamma_D, \\ \frac{\partial u}{\partial n} &= g \quad \text{on } \Gamma_N, \end{aligned}$$



where

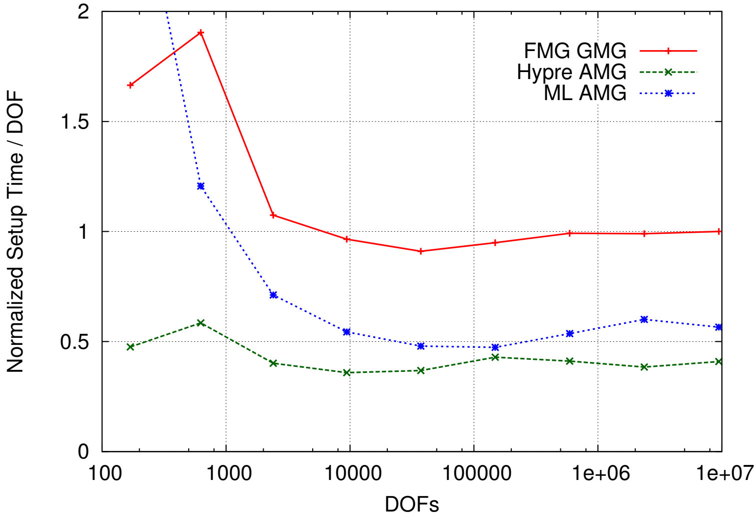
$$\begin{aligned} \Omega &= (0,1) \times (0,1), \\ \Gamma_D &= \{(x,y) \in \partial\Omega : x = 0 \vee x = 1\}, \\ \Gamma_N &= \partial\Omega \setminus \Gamma_D, \\ f(x,y) &= 10\exp(-((x-0.5)^2 + (y-0.5)^2)/0.02), \\ g(x,y) &= \sin(5x). \end{aligned}$$

Poisson Problem (2D)

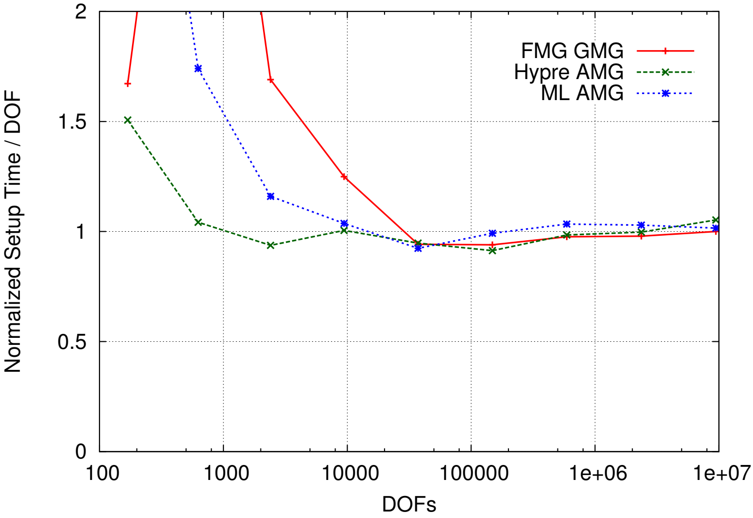
Test Parameters

- discretization: P1, P2, P3 and P4 Lagrange-Elements
- coarse grid: 12x12 (P1), 6x6 (P2), 4x4 (P3), 3x3 (P4) `UnitSquare`
- 8 refinements
- symmetric Gauss-Seidel smoother (SSOR with $\omega = 1$)
- V-cycle scheme
- termination criteria for PCG: $||C^{-1}r||/||C^{-1}b|| < 10^{-6}$

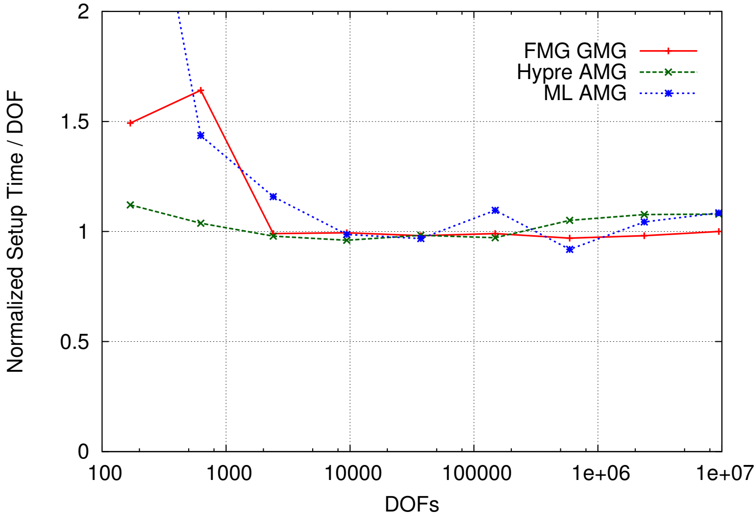
Normalized Setup Time / DOF (P1)



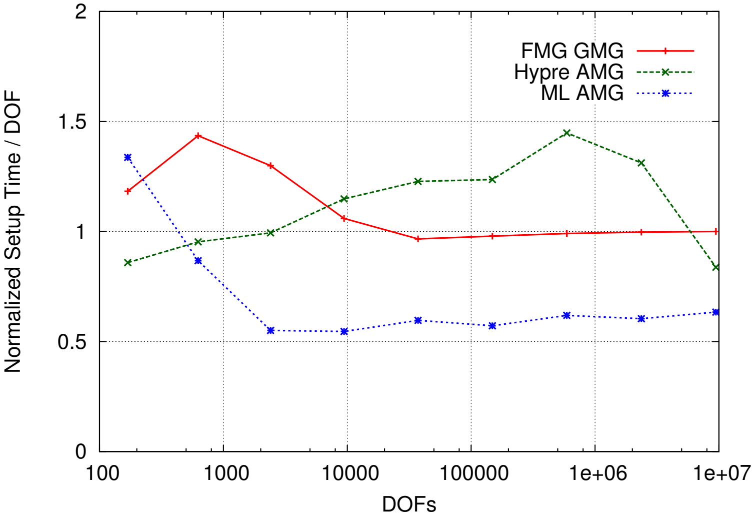
Normalized Setup Time / DOF (P2)



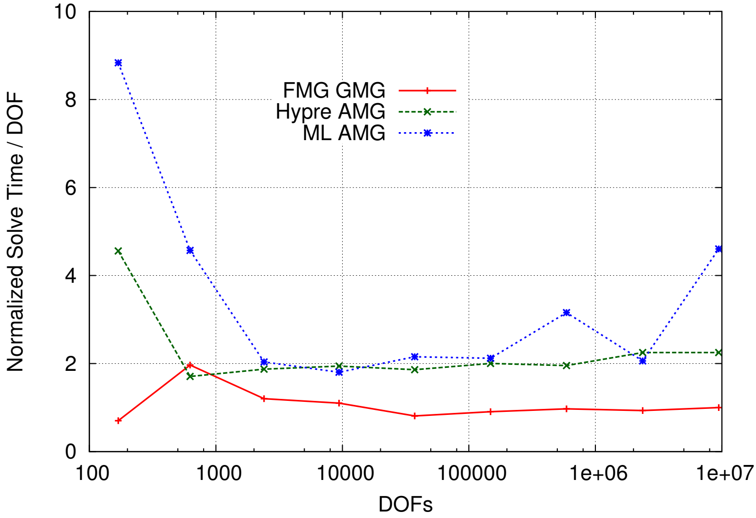
Normalized Setup Time / DOF (P3)



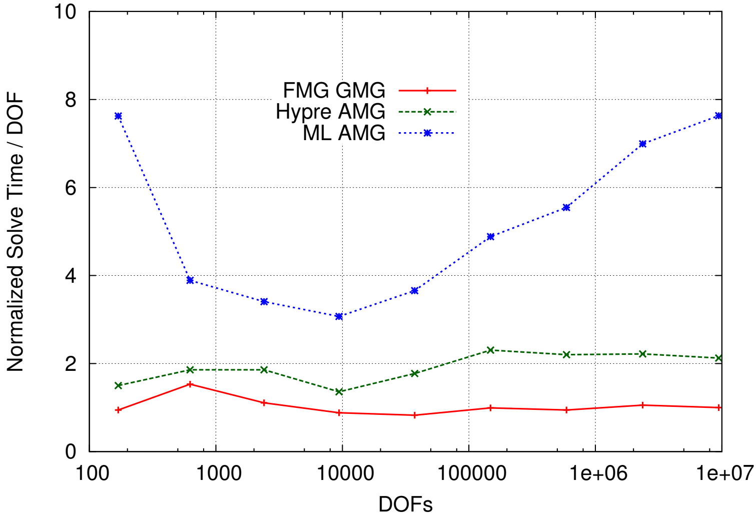
Normalized Setup Time / DOF (P4)



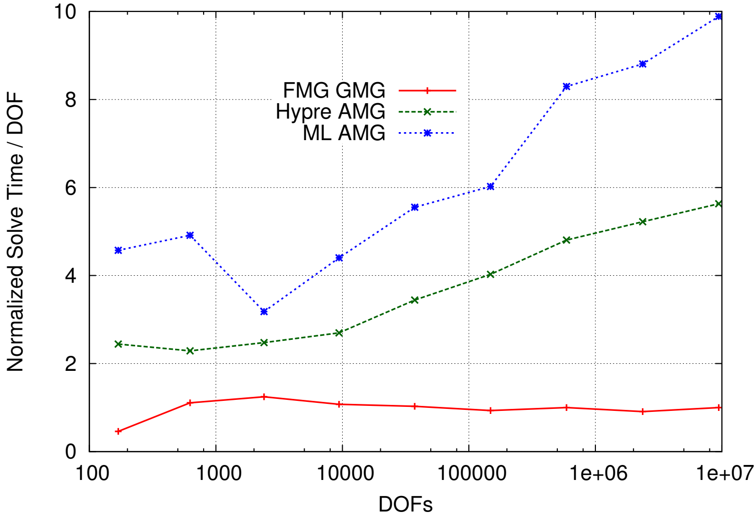
Normalized Solve Time / DOF (P1)



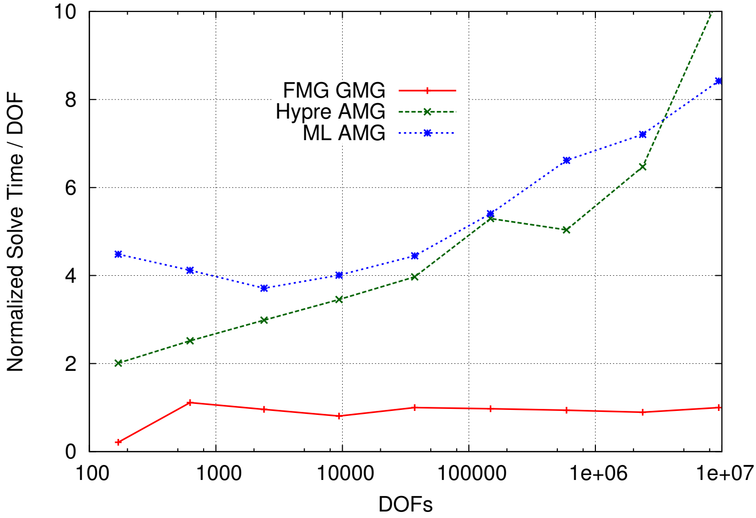
Normalized Solve Time / DOF (P2)



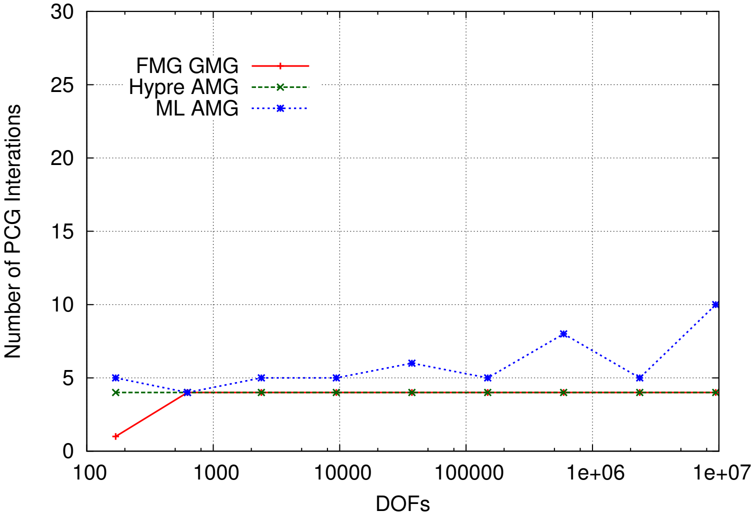
Normalized Solve Time / DOF (P3)



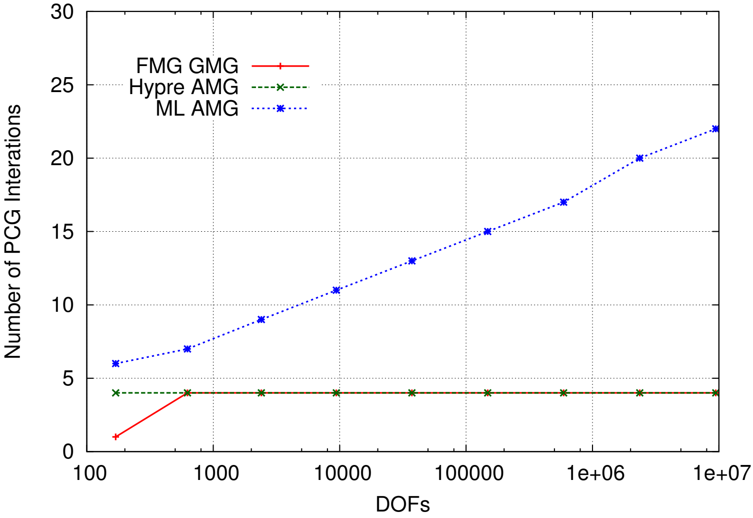
Normalized Solve Time / DOF (P4)



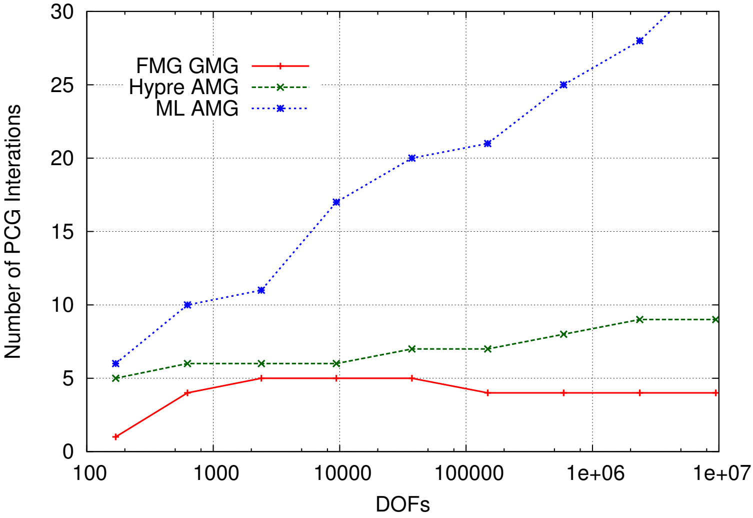
Number of PCG Iterations (P1)



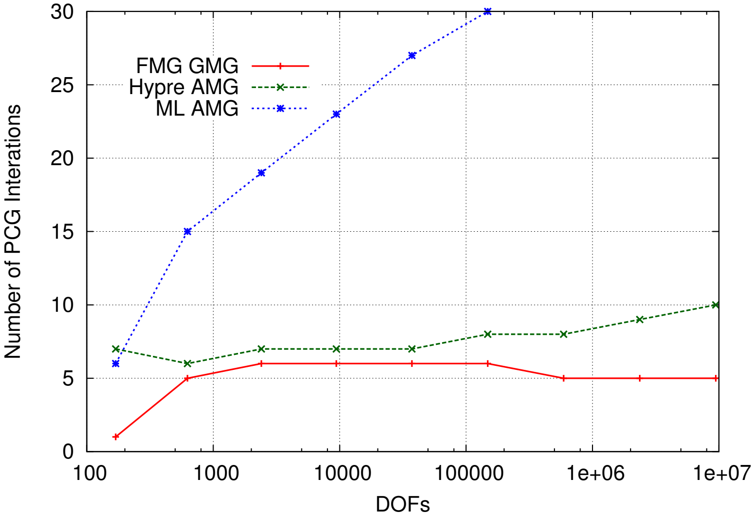
Number of PCG Iterations (P2)



Number of PCG Iterations (P3)



Number of PCG Iterations (P4)



Linear Elasticity (3D)

$$
\begin{aligned}
\operatorname{div}(\sigma(u)) + f &= 0 \quad \text{in } \Omega, \\
\sigma(u) \cdot n &= 0 \quad \text{on } \Gamma_N, \\
u &= 0 \quad \text{on } \Gamma_D,
\end{aligned}
$$

with isotropic material law

$$
\sigma(u) = 2\,\mu\,\epsilon(u) + \lambda\,\mathrm{tr}(\epsilon(u))\,I,
$$

and with the deformation

$$
\epsilon(u) = \frac{1}{2}\left(\mathrm{grad}(u) + \mathrm{grad}(u)^T\right).
$$

(λ, $\mu > 0$: Lamé parameters)

Linear Elasticity (3D)

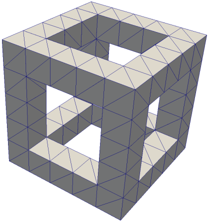
Test Parameters

$$\Omega = a^3 \setminus (a \times b^2 \cup b \times a \times b \cup b^2 \times a)$$

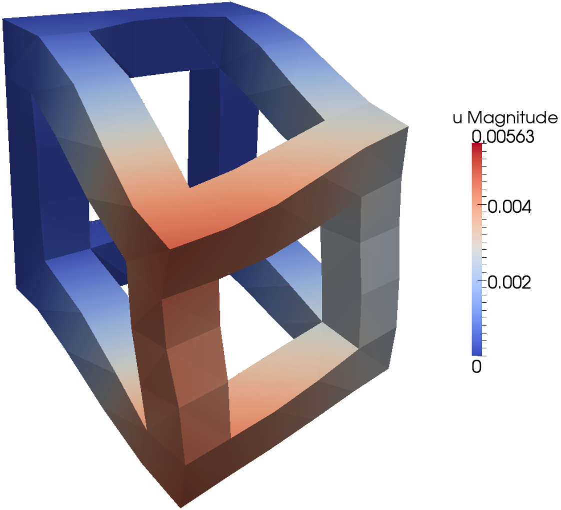
$$a = (0,1),\, b = (1/6, 5/6)$$

$$f(x,y,z) = \vec{e_z} \begin{cases} -10\exp(2z+y) & : x \geq 5/6 \\ 0 & : \text{else} \end{cases}$$

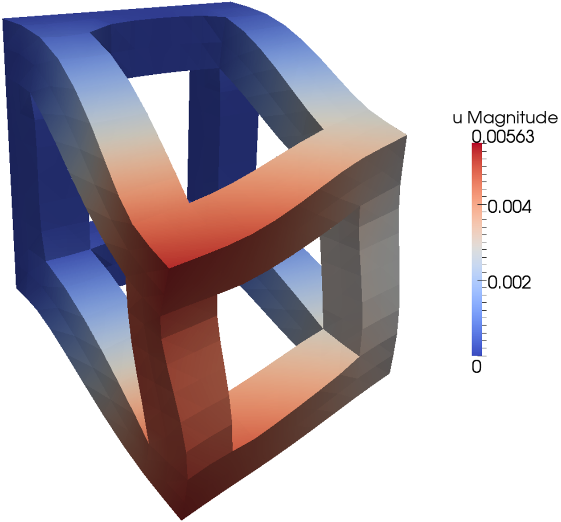
$$\Gamma_D = \{(x,y,z) \in \overline{\Omega} : x = 0\}$$



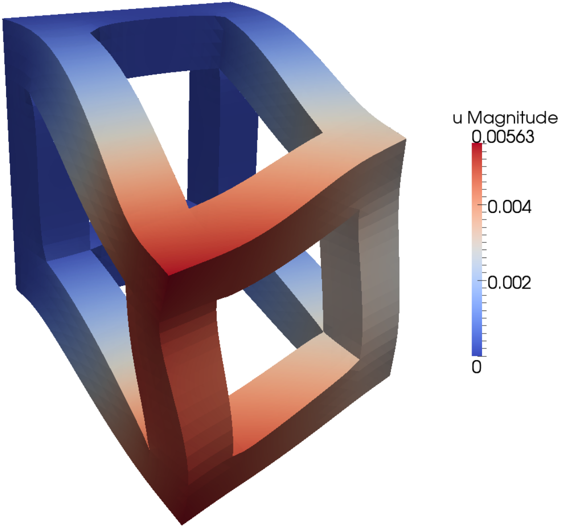
Deformation (Refinement 0)



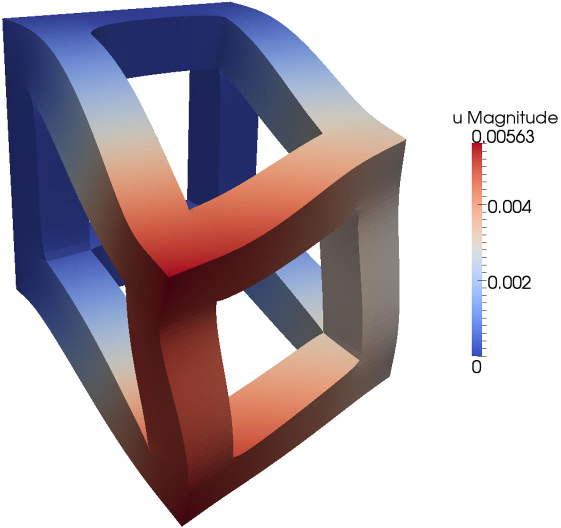
Deformation (Refinement 1)



Deformation (Refinement 2)



Deformation (Refinement 3)



Solve Time and Number of PCG Iterations (P2)

Refinements	DOFs	FMG GMG Iterations	FMG GMG Solve Time	Hypre AMG Iterations	Hypre AMG Solve Time	ML AMG Iterations	ML AMG Solve Time
0	2 268	1	0.0357	64	0.353	94	0.198
1	12 900	7	0.102	113	8.33	193	2.55
2	84 564	7	0.703	218	215.	398	50.5
3	606 900	8	6.11				
4	4 586 868	8	48.3				

AMG does not work well for this problem!?

Stokes Problem (2D)

$$
\begin{aligned}
-\mu \Delta u + \nabla p &= f \quad \text{in } \Omega, \\
\nabla \cdot u &= 0 \quad \text{in } \Omega, \\
u &= g \quad \text{on } \Gamma_D,
\end{aligned}
$$

results into the saddle point problem

$$
\begin{aligned}
a(u, v) + b(v, p) &= f(v) \quad \forall\, v \in H_0^1(\Omega)^2, \\
b(u, q) &= 0 \quad \forall\, q \in L^2(\Omega)/\mathbb{R},
\end{aligned}
$$

where

$$
\begin{aligned}
a(u, v) &= \mu \int_\Omega \nabla u : \nabla v \, \mathrm{d}x, \\
b(v, p) &= -\int_\Omega p \nabla \cdot v \, \mathrm{d}x, \quad f(v) = \int_\Omega f\, v \, \mathrm{d}x.
\end{aligned}
$$

Stokes Problem (2D)

linear system of equations (Taylor-Hood elements):

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

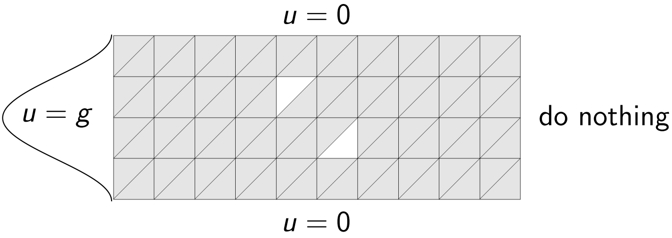
MINRES preconditioning:

$$C = \begin{pmatrix} A & 0 \\ 0 & M \end{pmatrix}$$

M: mass matrix for pressure

M-block: LU-decomposition (small matrix)
A-block: LU-decomposition, FMG GMG, Hypre AMG, ML AMG

Stokes Problem (2D)



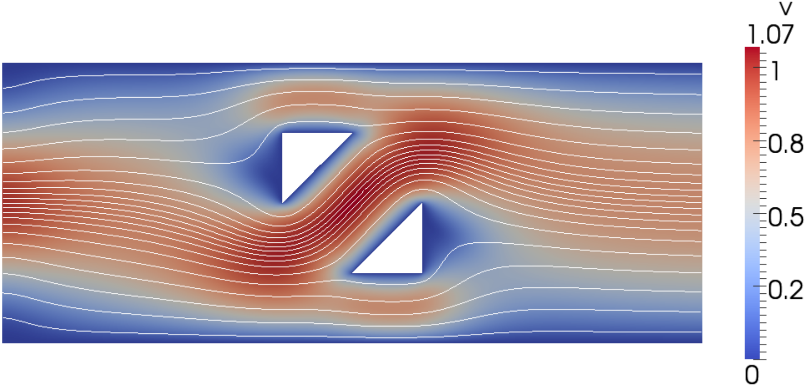
Test Parameters

$$\Omega = ((0,1) \times (0.3, 0.7)) \setminus (T_1 \cup T_2),$$

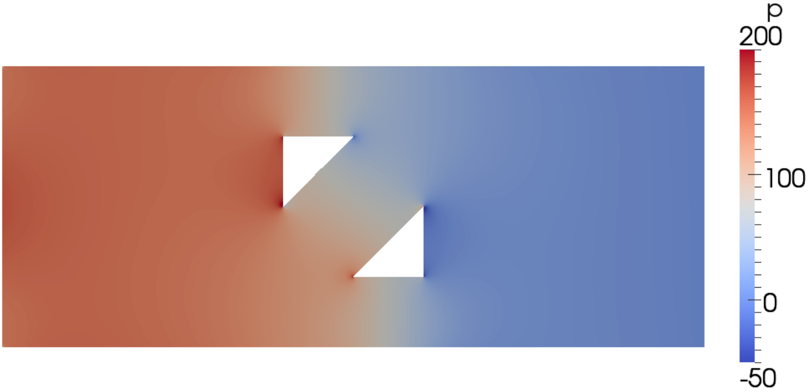
$$\Gamma_D = \partial\Omega \setminus (\{1\} \times (0.3, 0.7)),$$

$$g(x,y) = \begin{cases} (\cos^2(\pi(y-0.5)/0.4), 0)^T & \text{für } x = 0 \\ (0,0)^T & \text{sonst.} \end{cases}$$

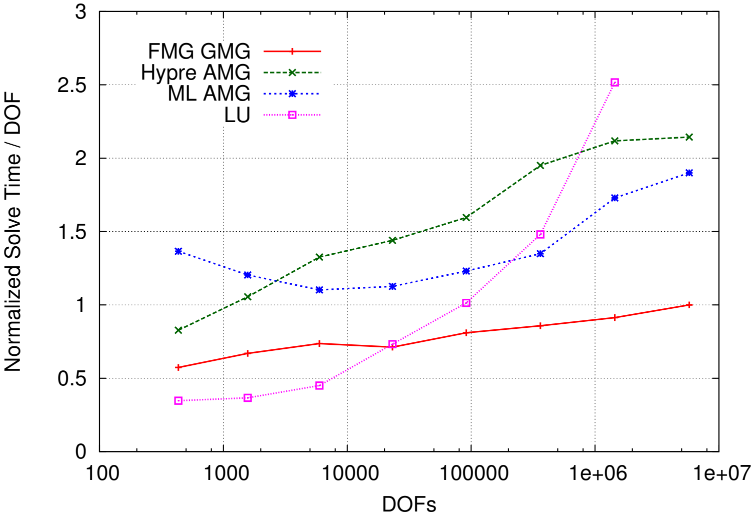
Solution (Velocity)



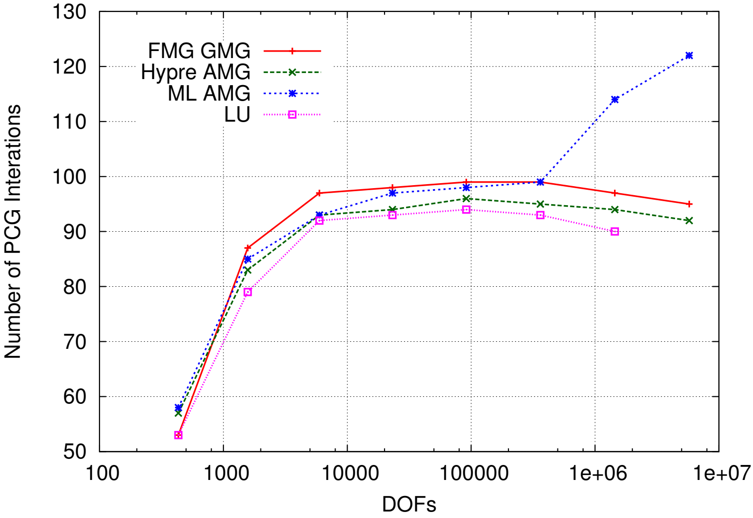
Solution (Pressure)



Normalized Solve Time / DOF



Number of PCG Iterations



Conclusion and Outlook

Conclusion and Outlook

GMG works very well.

Things to do:

- reduce the solver setup time,
- enable parallel computing (prolongation, restriction),
- Python support (SWIG),
- customized smoothers,
- block preconditioning,
- FAS (nonlinear) multigrid,
- demos with mesh adaptivity,
- ...

Thank you for your attention!

http://launchpad.net/fmg