November 3, 2017

Structured grid algorithms in MueLu

Luc Berger-Vergiat
Center for Computing Research

Sandia National Laboratories
Albuquerque, New Mexico USA

SAND 2017-XXXXX YY

Outline

Some notations

A linear system is denoted

$$Ax = b, \tag{1}$$

we assume that A has size $n \times n$, x is the solution and $\bar{x}$ the exact solution.
The residual vector is defined as

$$r = b - Ax. \tag{2}$$

Noticing that $b = A\bar{x}$ we get

$$r = A\bar{x} - Ax. \tag{3}$$

Which prompts us to introduce the error $e = \bar{x} - x$ and the correction equation

$$Ae = r. \tag{4}$$

Two types of solvers

1. Direct solvers: great for small size problems, also good for large banded problems, usually simple to use, do not scale (both complexity and memory), converge in 1 iteration
   - Gaussian elimination,
   - Factorization,
   - fast Poisson solvers based on FFT.
2. Iterative solvers: good for large problems, less sensitive to large band width, harder to use (more parameters), scale well, may take many iterations to converge
   - fixed point iterations, Jacobi, GS, ILU, masked LU...
   - Krylov solvers, CG, CGS, BICGSTAB, GMRES

Mixing it up

Iterative solvers' convergence can be accelerated using preconditioners

$$Ax = b \Leftrightarrow P^{-1}(Ax - b) = 0 \tag{5}$$

if P (the preconditioner) is not singular.
Desired properties for P

- cheap $P^{-1}b$ evaluation,
- smaller condition number for $P^{-1}A$,
- preserve good properties of A: symmetry, ellipticity...

Mixing it up

Iterative solvers' convergence can be accelerated using preconditioners

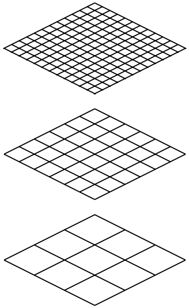
$$Ax = b \Leftrightarrow P^{-1}(Ax - b) = 0 \tag{5}$$

if P (the preconditioner) is not singular.
Desired properties for P

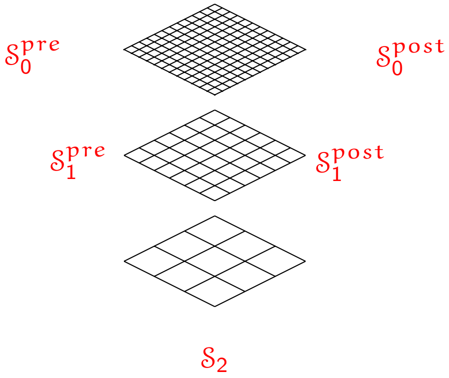
- cheap $P^{-1}b$ evaluation,
- smaller condition number for $P^{-1}A$,
- preserve good properties of A: symmetry, ellipticity...

For this presentation let us focus on my favorite P: the multigrid method

Multigrid method



Multigrid method

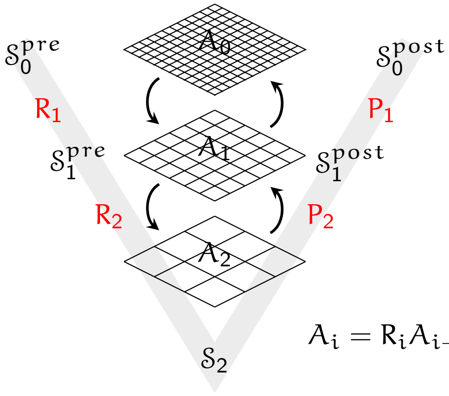


$\mathcal{S}_0^{\text{pre}}$
$\mathcal{S}_0^{\text{post}}$
$\mathcal{S}_1^{\text{pre}}$
$\mathcal{S}_1^{\text{post}}$
$\mathcal{S}_2$

Two main components

- Smoothers
  - "Cheap" reduction of oscillatory error (high energy)
  - $\mathcal{S}_L \approx A_L^{-1}$ on the coarsest level L

Multigrid method



$$A_i = R_i A_{i-1} P_i$$

Two main components

- Smoothers
  - "Cheap" reduction of oscillatory error (high energy)
  - $\mathcal{S}_L \approx A_L^{-1}$ on the coarsest level L
- Grid transfers (prolongators and restrictors)
  - Definition of coarse level matrices (setup phase)
  - Data movement between levels (solve phase).

Classic smoothers

Split the numerical operator

$$A = M - N \tag{6}$$

taking care of choosing an M easily invertible.
Use a fixed point iteration based on the above split

$$Mx_{n+1} = Nx_n + b, \tag{7}$$

additionally some damping may be introduced:

$$x_{n+1} = \omega M^{-1}(Nx_n + b) + (1 - \omega)x_n. \tag{8}$$

Jacobi's method

The simplest split: $M = diag(A)$ leads to the Jacobi iteration.

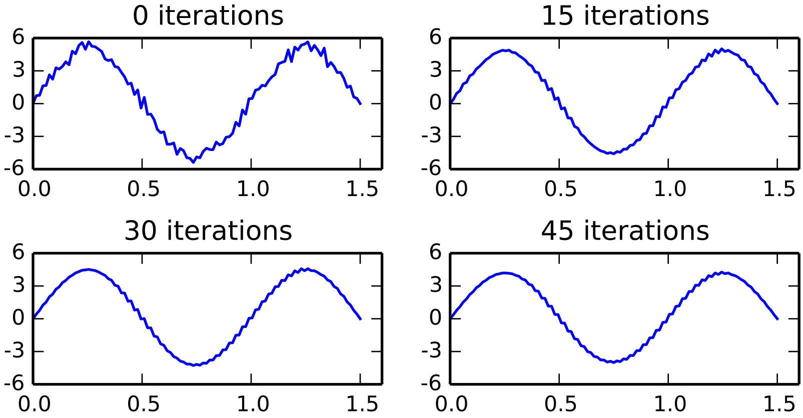


Figure: Effect of Jacobi iterations on the error

Damped Jacobi's method

If oscillation persist between two consecutive grid points: try damping!

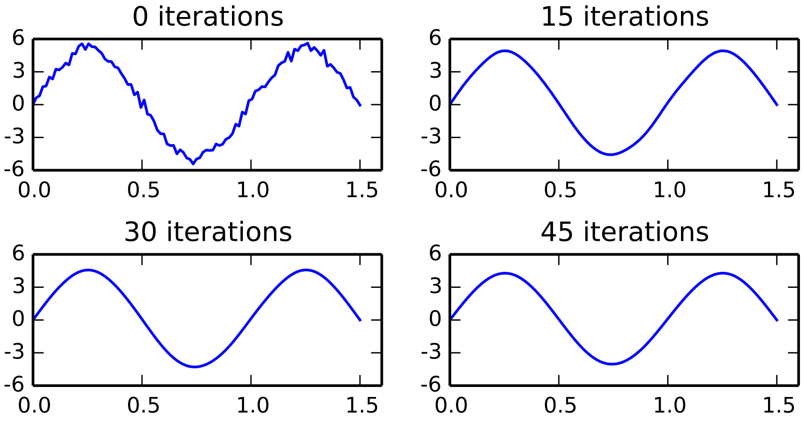


Figure: Effect of Jacobi iterations on the error

Smoothing 2D problems

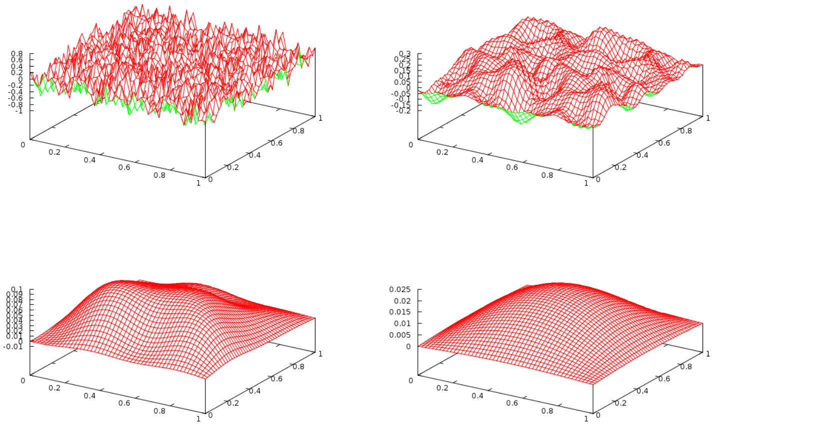
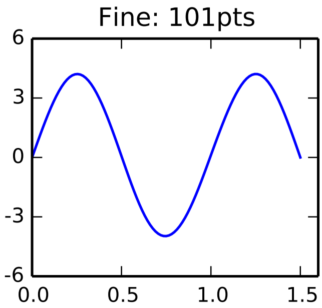


Figure: Damped Jacobi in 2D: images courtesy of the MueLu tutorial

The effect of coarsening

Jacobi iterations remove high frequencies quickly but stall on low frequencies. This happens to most smoothers...
The smooth error is well represented with much fewer grid points!



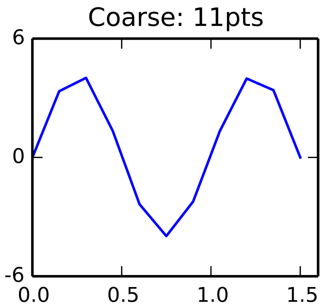


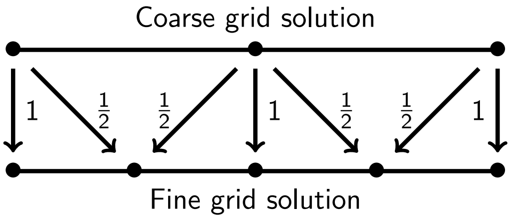
Figure: Representation of the error on two grids

Now smooth coarse problem or solve with LU if small enough.

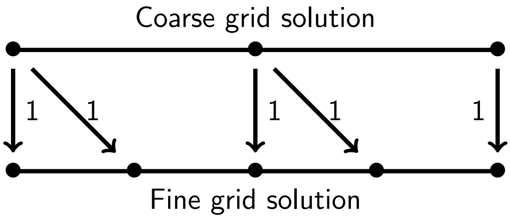
Simple coarsening scheme

Going from coarse grid to fine grid is easily achieved using:

- Linear interpolation



- Piece-wise constant:



Galerkin projection of A

Linear case:

$$A_h = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$A_H = P^t A_f P = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \tag{10}$$

Galerkin projection of A

Piece-wise constant case:

$$A_h = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$$A_H = P^t A_f P = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \tag{12}$$

Multigrid properties

Mutligrid

- solves/accelerates the solution of linear problems,
- has a theoretical complexity $\mathrm{O}(\mathrm{N})$, with N the $\#$ unknowns,
- can be implemented in parallel but requires a lot of programing effort,
- comes in two main flavors: Geometric and Algebraic.

For Poisson's equation discretized on a uniform grid you can expect the number of iterations until convergence to remain constant for varying mesh sizes.

Convergence might depend on mesh size for more complex equations/discretizations.

Outline

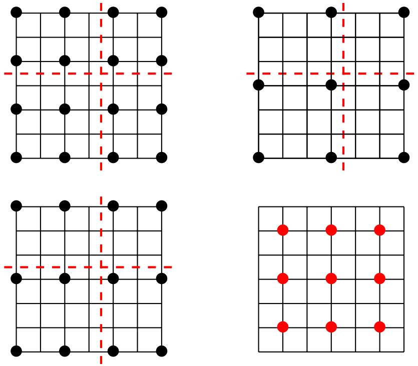
What is MueLu?

MueLu:

- is the multigrid package succeeding ML within Trilinos[1],
- provides an extendable set of grid transfer and smoother algorithms,
- is programmed in C++11 and relies heavily on polymorphism via templating (compile time polymorphism) and object oriented progamming (run time polymorphism),
- is available on multiple platforms: Linux, OsX, Windows, ...
- and runs on multiple architectures: multiple cores, many cores, GPU,
- can use 64bits indexing to solve problems with more than 4B equations.

[1] The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems.

Structured grid coarsening



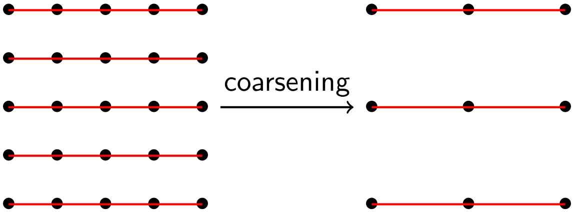
Red dashed lines - -, represent processor boundaries.

1. Coarsening rates are variable and independent in each direction,
2. coarse points are chosen to include boundary points (better for BC),
3. coarsening is continuous across processor boundary for uniform coarse point distribution
4. coarsening rate is automatically reduced at mesh boundary.

Structured line detection

Structured grid coarsening allows for line/plane detection on each grid level allowing for:

- semi-coarsening with plane relaxation
- line smoothing on anisotropic mesh/problem
- variable coarsening on anisotropic meshes



Interpolation grid transfer

Both piece-wise constant and linear interpolation between coarse points are available.

Linear interpolation:

- relies on Newton iteration to find interpolation weights (more fragile),
- takes geometric distances into account (requires coordinates).

Piece-wise constant interpolation:

- uses (i,j,k) indexes for coarsening,
- distance based variant relies on coordinates (not tested so far),
- produces less fill on coarse grids.

Black box grid transfer

Black box multigrid is an algorithm designed to take advantage of the actual operator to compute the grid transfer operators. I was first proposed by Dendy [2, 3, 4] This algorithm relies on two ideas:

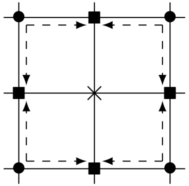
1. Schur complement provides "perfect" grid transfer operators

$$A_h = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix},\ P = \begin{bmatrix} -A_{ff}^{-1} A_{fc} \\ I_c \end{bmatrix},\ A_c = A_{cc} - A_{cf} A_{ff}^{-1} A_{fc}$$

2. dimensional decoupling generates sparsity and triangular structure

Black box grid transfer

Macro element schematic representation:



● : corner points

■ : edge points

× : interior points

Reorder nodes by type: interior, edge, corner

$$
\begin{array}{c}
c_1 \\ \gamma_1 \\ c_2 \\ \gamma_4 \\ \iota \\ \gamma_2 \\ c_4 \\ \gamma_3 \\ c_3
\end{array}
\begin{bmatrix}
\bullet & \bullet & & \bullet & \bullet & & & & \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & & & \\
 & & \bullet & \bullet & & \bullet & \bullet & & \\
\bullet & \bullet & & \bullet & \bullet & & \bullet & \bullet & \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
 & \bullet & \bullet & & \bullet & \bullet & & \bullet & \bullet \\
 & & & \bullet & \bullet & & \bullet & \bullet & \\
 & & & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
 & & & & \bullet & \bullet & & \bullet & \bullet
\end{bmatrix}
\rightarrow
\begin{array}{c}
\iota \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ c_1 \\ c_2 \\ c_3 \\ c_4
\end{array}
\left[\begin{array}{ccccc:cccc}
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & & \\
\bullet & \bullet & \bullet & \bullet & & & \bullet & \bullet & \\
\bullet & & \bullet & \bullet & \bullet & & & \bullet & \bullet \\
\bullet & \bullet & & \bullet & \bullet & \bullet & & & \bullet \\
\hdashline
\bullet & \bullet & & & \bullet & \bullet & & & \\
\bullet & \bullet & \bullet & & & & \bullet & & \\
\bullet & & \bullet & \bullet & & & & \bullet & \\
\bullet & & & \bullet & \bullet & & & & \bullet
\end{array}\right]
\tag{13}
$$

Black box grid transfer

Collapse stencils on edge, i.e. for a vertical, respectively horizontal edge:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \to \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}; \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \to \begin{bmatrix} -1 & 2 & -1 \end{bmatrix},$$

which leads to the following sparsity pattern:

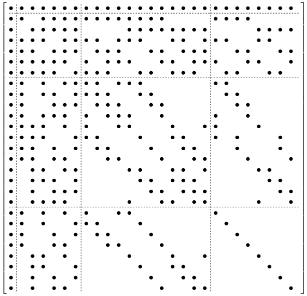
$$\begin{array}{c} \iota \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}\left[\begin{array}{ccccc:cccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & & \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & & & \bullet & \bullet & \\ \bullet & & \bullet & \bullet & \bullet & & & \bullet & \bullet \\ \bullet & \bullet & & \bullet & \bullet & \bullet & & & \bullet \\ \hdashline \bullet & \bullet & & & \bullet & \bullet & & & \\ \bullet & \bullet & \bullet & & & & \bullet & & \\ \bullet & & \bullet & \bullet & & & & \bullet & \\ \bullet & & & \bullet & \bullet & & & & \bullet \end{array}\right] \to \begin{array}{c} \iota \\ \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}\left[\begin{array}{ccccc:cccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ & \bullet & & & & \bullet & \bullet & & \\ & & \bullet & & & & \bullet & \bullet & \\ & & & \bullet & & & & \bullet & \bullet \\ & & & & \bullet & \bullet & & & \bullet \\ \hdashline \bullet & \bullet & & & \bullet & \bullet & & & \\ \bullet & \bullet & \bullet & & & & \bullet & & \\ \bullet & & \bullet & \bullet & & & & \bullet & \\ \bullet & & & \bullet & \bullet & & & & \bullet \end{array}\right] = \begin{bmatrix} \hat{A}_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}$$

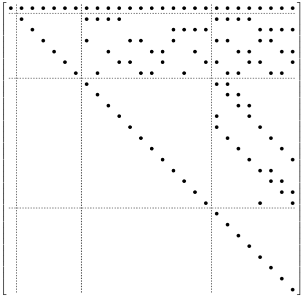
Black box grid transfer

Now $\hat{A}_{ff}$ is readily invertible:

$$\hat{A}_{ff} = \left[\begin{array}{c|cccc} \bullet & \bullet & \bullet & \bullet & \bullet \\ \hline & \bullet & & & \\ & & \bullet & & \\ & & & \bullet & \\ & & & & \bullet \end{array}\right] = \begin{bmatrix} A_{\iota\iota} & A_{\iota\gamma} \\ 0 & \hat{A}_{\gamma\gamma} \end{bmatrix} \to \hat{A}_{ff}^{-1} = -\begin{bmatrix} A_{\iota\iota}^{-1} & -A_{\iota\iota}^{-1} A_{\iota\gamma} \hat{A}_{\gamma\gamma}^{-1} \\ 0 & \hat{A}_{\gamma\gamma}^{-1} \end{bmatrix}$$

The same ideas also apply to 3D problem:





Outline

Poisson on a cube

Start with an easy problem: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f$

Using the exact solution:
$u(x, y, z) = \sin(\pi x)\sin(\pi y)\sin(\pi z)\exp(x + y + z)$ to compute the forcing term.

In all the following example we use a geometric multigrid (GMG) algorithm with a single sweep of damped (0.9) Jacobi pre- and post-smoother and an LU coarse grid solver.

Serial experimentations

For serial examples 3D uniform grids are used and FEM is chosen as the discretization method, the grid transfer operators are constructed using linear interpolation (piece-wise constant gives same results for Poisson).

Iterations	Grid levels			
	2	3	4	5
1	0.0062253	0.006397	0.006397	0.006397
2	4.4829e-05	4.6156e-05	4.6156e-05	4.6156e-05
3	3.4685e-07	3.5036e-07	3.5036e-07	3.5036e-07
4	2.4203e-09	2.4898e-09	2.4898e-09	2.4898e-09
5	1.5014e-11	1.6925e-11	1.6925e-11	1.6925e-11
6	9.9889e-14	1.262e-13	1.2619e-13	1.262e-13
7	2.3459e-15	2.4433e-15	2.4363e-15	2.4445e-15

Table: Using GMG as fixed point iteration solver with varying number of levels on a mesh with 4913 points (17^3).

Serial experimentations

Same problem using varying coarsening rate

Iterations	Coarsening rate 2	3	4	5
1	5.3674e-04	1.2750e-03	1.8536e-03	2.0950e-03
2	6.2792e-06	4.3705e-05	1.9476e-04	2.3626e-04
3	1.2070e-07	2.1998e-06	2.8050e-05	4.3682e-05
4	2.6778e-09	1.2962e-07	4.5258e-06	9.7886e-06
5	6.0525e-11	8.4691e-09	7.7184e-07	2.4465e-06
6	1.3386e-12	5.8442e-10	1.3501e-07	6.5032e-07
7	3.0480e-14	4.1147e-11	2.3897e-08	1.7805e-07
8	7.9183e-16	2.9033e-12	4.2500e-09	4.9413e-08
9	1.0104e-16	2.0357e-13	7.5659e-10	1.3802e-08
10	9.2781e-17	1.4138e-14	1.3454e-10	3.8693e-09

Table: The finest grid is discretized with $n = (1 + c^2)^3$ points, where c is the coarsening rate, two grid levels are used and the coarse grid contains 8 points.

Parallel resulst

Here we use GMG as a preconditioner for a GMRES linear solver, the coarsening rate is set to 2 and the coarse grid contains 10 or less points. The convergence criteria for these simulations is $\frac{||R||}{||R_0||} \leqslant 10^{15}$.

MPI ranks	Mesh size			
	8,000	64,000	512,000	4,096,000
1	6	7	7	7
2	7	7	7	7
4	7	7	7	7
8	7	7	8	8

Table: Parallel behavior of the multigrid algorithm.

Note: one could use CG instead of GMRES for Poisson but GMRES is our default solver.

Thermal Navier-Stokes flow

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_i(\mathbf{U})}{\partial x_i} - \frac{\partial \mathbf{G}_i(\mathbf{U})}{\partial x_i} = \mathbf{0} \tag{14}$$

with

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho v_j \\ \rho E \end{pmatrix}, \ \mathbf{F}_i(\mathbf{U}) = \begin{pmatrix} \rho v_i \\ \rho v_i v_j + P\delta_{ij} \\ \rho E v_i + P v_i \end{pmatrix} \text{ and } \mathbf{G}_i(\mathbf{U}) = \begin{pmatrix} 0 \\ \tau_{ij} \\ \tau_{ij} v_j - q_i \end{pmatrix} \tag{15}$$

where ρ is the fluid density, v is the fluid velocity and E the fluid energy per unit of mass which is expressed as $E = \frac{1}{2} v_i v_i + e$ the sum of the kinetic and internal energy e. P is the fluid pressure, τ_{ij} is the viscous stress tensor. $q_i = -\kappa \frac{\partial T}{\partial x_i}$ is the heat flux, T the temperature and κ the thermal conductivity of the gas.

Thermal Navier-Stokes flow

Sandia National Laboratories

For a Newtonian fluid (linear stress/strain) the stress can be expressed as

$$\tau_{ij} = \mu \left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) + \lambda \delta_{ij} \left(\frac{\partial v_k}{\partial x_k} \right) \tag{16}$$

with μ the viscosity and λ the bulk viscosity (often $\lambda = -\frac{2}{3}\mu$ for Newtonian fluid). Finally the pressure is given by the equation of state (EOS) of the fluid. For a perfect gas we have

$$P = \rho R T \tag{17}$$

with R perfect gas constant. More details on the mechanical formulation used to represent the fluid behavior can be found in [5] and [6].

Time and spacial discretization

1. The time integration is performed using an implicit Euler scheme
2. Non-linear problem resulting from the time integration are inexactly solved with 1 Newton iteration.
3. The system is discretized using a cell-centered finite volume scheme [5], stabilization is added to the operator with a SUPG formulation for the fluid pressure [1].

We are interested in a steady state problems so a pseudo-time integration scheme is used to ramp-up the CFL number associated with the time integration problem (target CFL range: 1K$\sim$10K).

Blunt wedge problem

The mesh is structured and conatins 144^3 cells, the input flow is supersonic: Mach 3.
The linear interpolation scheme is not stable (iterations increase with number of levels), piece-wise constant interpolation is used.
The number of equations per level is aggressively reduce using a coarsening rate of 3.
A unique sweep of pre-smoother is employed to keep iteration costs low, two smoothers are used:

1. additive Schwarz with one level of overlap and and ILU(0) solves on each subdomain using line preserving partitions,
2. line block Jacobi smoother using line preserving partitions (lines follow the cross flow direction).

Serial experiment

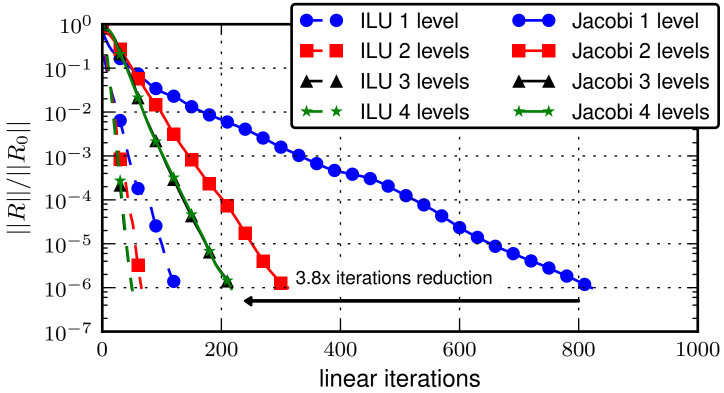


Figure: Unstable coarse operators forbid using to many levels and a direct solver on coarsest grid.

Parallel experiement

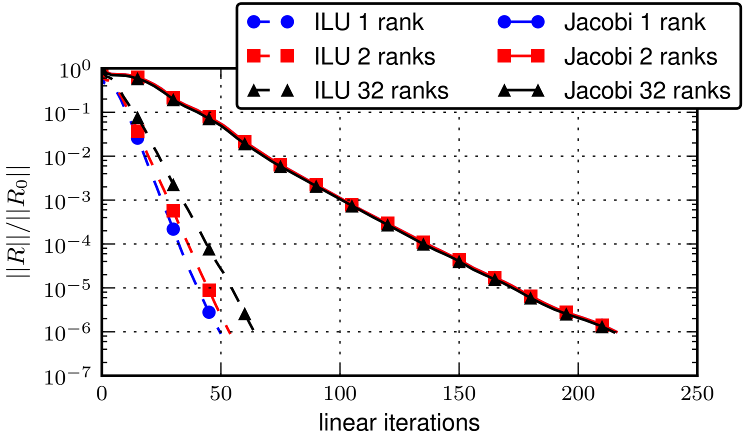


Figure: Due to line preserving partitioning the performance of the algorithm using the line block Jacobi smoother is independent of the number of processors used.

Damping of the preconditioner

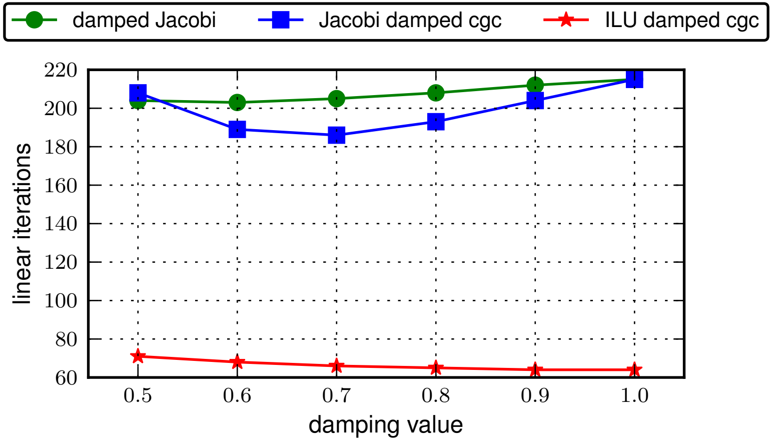


Figure: Interestingly damping the coarse grid correction is more effective than damping the preconditioner. This is indicative of instability in the coarse operators: $(\mathrm{I} - \alpha \mathrm{D}^{-1}\mathrm{A})$ vs. $\alpha(\mathrm{I} - \mathrm{D}^{-1}\mathrm{A})$.

Outline

Concluding remarks

We have achieved:

- the implementation of a parallel geometric multigrid algorithm in MueLu using variable coarsening rate and line smoothing on all levels,
- said algorithm performance is tested on simple Poisson 3D problem in serial and parallel,
- finally initial experiments are conducted on the blunt wedge problem to assess the performance of the preconditioner on a problem of interest.

Concluding remarks

We are working on:

- the implementation of a black box multigrid algorithm with variable coarsening rate: code generates correct grid transfer operators on simple meshes, more tests are needed before production runs,
- both geometric and black box algorithms are extended to block meshes,
- the algorithm need to be tested on more challenging CFD problems: complex geometries, higher velocities, reacting gas problem, etc...

References

R.D. Falgout, *An Introduction to Algebraic Multigrid*, in Computing in Science & Engineering Volume 8, Issue 6 (2006) 24–33, 10.1109/MCSE.2006.105

J.E. Dendy, *Black box multigrid*, Journal of Computational Physics, 1982 Dec 1;48(3):366-86.

J.E. Dendy *Black box multigrid for nonsymmetric problems*, Applied Mathematics and Computation. 1983 Jan 1;13(3-4):261-83.

J.D. Moulton *Black Box Multigrid with coarsening by a factor of three*, Numerical Linear Algebra with Applications. 2010 Apr 1;17(2?3):577-98.

M. Howard, A. Bradley, S.W. Bova, J. Overfelt, R. Wagnild, D. Dinzl, M. Hoemmen, A. Klinvex *Towards Performance Portability in a Compressible CFD Code*, 23rd AIAA Computational Fluid Dynamics Conference. Denver, Colorado.

F. Blottner, *Accurate Navier-Stokes results for the hypersonic flow over a spherical nosetip* Journal of spacecraft and Rockets. 1990 Mar 1;27(2):113-22.

References

P.B. Bochev, M.D. Gunzburger, J.N. Shadid, *Stability of the SUPG finite element method for transient advection?diffusion problems*, in Computer Methods in Applied Mechanics and Engineering, Volume 193, Issues 23–26, 2004, Pages 2301-2323