Algebraic multigrid in PETSc

Mark Adams
Lawrence Berkeley National Laboratory

PETSc user meeting, 29 June 2016

Outline

- High level discussion of multilevel iterative solver support in PETSc
- Algebraic multigrid (AMG) introduction
- PETSc's native AMG framework – GAMG
- Performance debugging GAMG

Start with bigger problem in PETSc

- How to get users, at all levels of expertise and capacity, to where they should be
  - "It takes considerable time for us to answer these questions, and so it would be better to consult the documentation and published reports first before using the mailing list heavily.", Matt Knepley (Sunday)
  - More discussion on this today (I think)
- There seems to be something about human psychology that makes people think that PETSc solver should just work (fast, low memory, and accurate) out of the box
  - Maybe because it is so conceptually simple: Ax=b
  - Scalable [multilevel iterative] solvers for non-trivial PDEs are HARD
- PETSc's AMG documentation (written by me) is not great (Barry is not impressed at least), but a place to start
  - I'm not sure how to fix it
  - Writing a software paper with Toby Isaac and Garth Wells on GAMG
- I'm not going to talk about Domain Decomposition solvers in PETsc, but they share the same basic issues – multilevel iterative methods are hard

AMG not the only solver

- Multilevel iterative solvers are built to be scalable and fast.
  - This is a hard problem, multigrid is like a Ferrari:
    - Beautiful but finicky and not the best car for all situations
- Companies and laboratories spend a lot of money on solvers
  - You are lucky if it works, assume they will not
- Alternatives you should keep in mind and even start with:
  - Direct solvers: robust but not scalable (memory or time)
  - Simple iterative solvers (eg, ILU, default PETSc PC)
    - Not scalable in time
- Start with a small test, use a direct solver if need be, and get a good solution.
  - Then scale up - homotopy

OK, so you still want to use AMG

- GAMG: PETSc's native AMG (*-pc_type gamg*)
- Two good 3rd party solvers: *hypre* and *ML*
  - Hypre and ML are good solvers, may decades of work
    - I would recommend starting with hypre if avaialble
  - GAMG has about 3 man years
    - but I had a lot of experience before I started
    - Designed to be extensible by new contributors
  - Native solver supports complex, 64 bit integers and performance tools are more integrated
- AMG has lots of variables and variability
  - Test multiple AMG solvers: quick and easy.
- First, are you sure you want to use AMG …

Equations, discretizations, regimes

- What equations do you have?
  - Do you really have …
- Principal Component Analysis
  - Write matrix L scalar equations
    - Using Backward Euler for time
  - Det(L): Principal Components
  - Stokes falls out like this

$$\frac{d}{dt}u - \frac{d}{dx}v = 0$$

$$\frac{d}{dt}v - \frac{d}{dx}u = 0$$

$$L = \begin{pmatrix} 1 & -\Delta t\delta \\ -\Delta t\delta & 1 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

$$\det(L) = 1 - \Delta t^2 \delta^2$$

- Multigrid is great on elliptic operators
  - But special methods required if not “born elliptic”
- Stokes is common, so we have special things

Hyperbolic problems

- Hyperbolic problems are tricky
  - Mass term makes it simple as Δt goes to down to CFL (Courant–Friedrichs–Lewy) condition, (eg, explicit)
- Multigrid for computational fluid dynamics (CFD)
  - Active in 70's & 80's, using geometric multigrid
  - Using explicit time integrators as smoothers
  - Defect correction: Solve $L_1 x^{k+1} = f - L_2 x^k + L_1 x^k$
    - With stable low order L_1 and real operator L_2
- Ad: PETSc is developing new fast elliptic and hyperbolic solvers in mesh/discretization/solver framework DMPlex
  - With Toby Isaac and Matt
  - Includes p4est DM: DMForest ... back to AMG

Distributive Gauss-Seidel

- If Point G-S smoothing not sufficient principle component analysis
- Consider hyperbolic example:
- Use backward Euler,
- Now use right preconditioning
  - Distributed G-S solve Lu=f
- Define q by u = Cq
- Write update form of point-wise smoother B for LC:
  - $q \leftarrow q + B_{LC}(f - LCq)$
- Pre-multiply by C
  - $Cq \leftarrow Cq + CB_{LC}(f - LCq)$
- Use u = Cq
  - $u \leftarrow u + CB_{LC}(f - Lu)$
- C "distributes" update $B_{LC}(f - Lu)$
- Schur complement solvers, physics based PCs, are kind of similar

$$\frac{\partial u}{\partial t} - B \bullet \nabla w = 0$$

$$\frac{\partial w}{\partial t} - B \bullet \nabla u = 0$$

$$LU_{k+1} \equiv \begin{pmatrix} 1 & -\Delta t D \\ -\Delta t D & 1 \end{pmatrix} \begin{pmatrix} u_{k+1} \\ w_{k+1} \end{pmatrix} = \begin{pmatrix} u_k \\ w_k \end{pmatrix}$$

$$LC = \begin{pmatrix} 1 - \Delta t^2 D^2 & 0 \\ 0 & 1 - \Delta t^2 D^2 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & \Delta t D \\ \Delta t D & 1 \end{pmatrix}$$

So you really want AMG

- Discretizations are just as important as equations
  - Bad discretization of nice equations can kill you
- Accuracy and AMG performance often correlated
- You are not solving equations
  - You are solving discretized equations
- Example:
- First order term might be central differencing D^c
  - 2nd order might be D^+D^- because
  - D^cD^c is not stable
- This actually like defect correction
  - Use low order to precondition high order
  - A good idea, still, but be aware:
    - Can not reduce algebraic error to zero

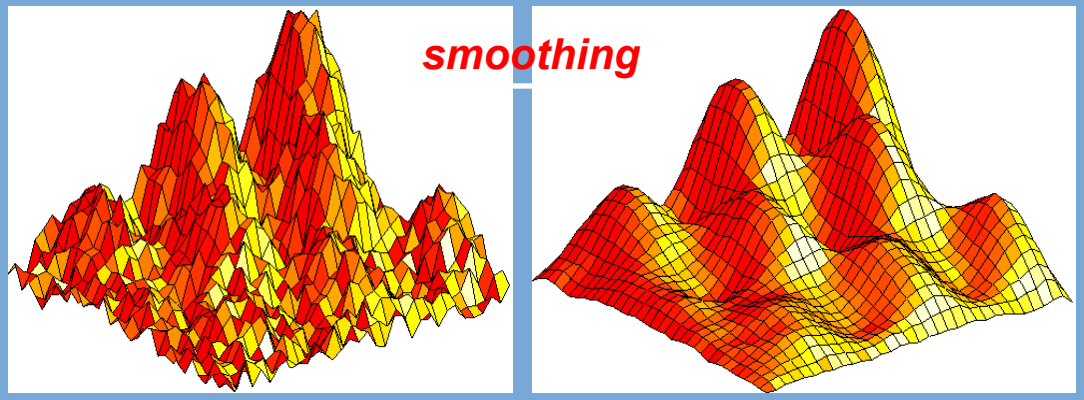
$$L = \begin{pmatrix} 1 & -\Delta t\delta \\ -\Delta t\delta & 1 \end{pmatrix}$$

$$\det(L) = 1 - \Delta t^2\delta^2$$

Outline

- High level discussion of multilevel iterative solver support in PETSc
- Algebraic multigrid (AMG) introduction
- PETSc's native AMG framework – GAMG
- Performance debugging GAMG

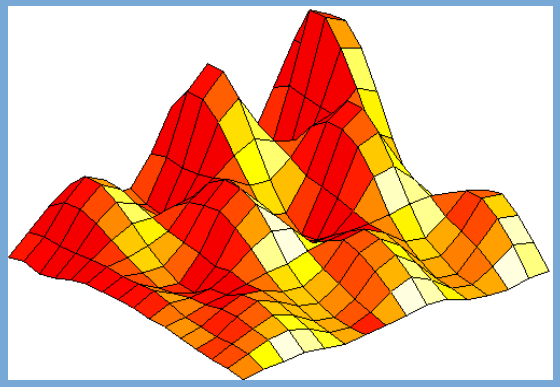
Fine, you want AMG …. here goes



Finest Grid

The Multigrid V-cycle

Restriction (R)



First Coarse Grid

Note: smaller grid

Prolongation (P) (interpolation)

Multigrid V(ν_1,ν_2) & F(ν_1,ν_2) cycle

- function u = MGV(A,f)
  - If A coarsest grid
    - $u \leftarrow A^{-1}f$
  - else
    - $u \leftarrow S^{\nu 1}(f, u)$ -- Smoother (pre)
    - $r_H \leftarrow R(f - Au)$
    - $e_H \leftarrow$ MGV(RAP, r_H) -- recursion (Galerkin)
    - $u \leftarrow u + Pe_H$
    - $u \leftarrow S^{\nu 2}(f, u)$ -- Smoother (post)
- function u = MGF(A_i,f)
  - if A_i is coarsest grid
    - $u \leftarrow A_i^{-1}f$
  - else
    - $r_H \leftarrow R(f - A_i u)$
    - $e_H \leftarrow$ FGV(A_{i+1}, r_H) -- recursion
    - $u \leftarrow Pe_H$
    - $u \leftarrow u +$ MGV(A_i, $f - A_i u$)

Algebraic Multigrid (AMG) methods

- MG Coarse grid spaces (columns of P) define MG method
  - Geometric MG: Finite element spaces – requires FE mesh
  - Algebraic MG: Generalize and induce from operator
- MG Smoothers: Additive and multiplicative
  - Multiplicative: good theoretical properties, parallelism hard
  - Additive requires damping (eg, Chebyshev polynomials)
  - Smoothers are important for problem specific tuning and overall performance
- Given:
  - P ($R=P^T$) and smoother S
  - Galerkin coarse grids: $A_{i+1} = R A_i P$
  - A multigrid algorithm essentially fully defined with P

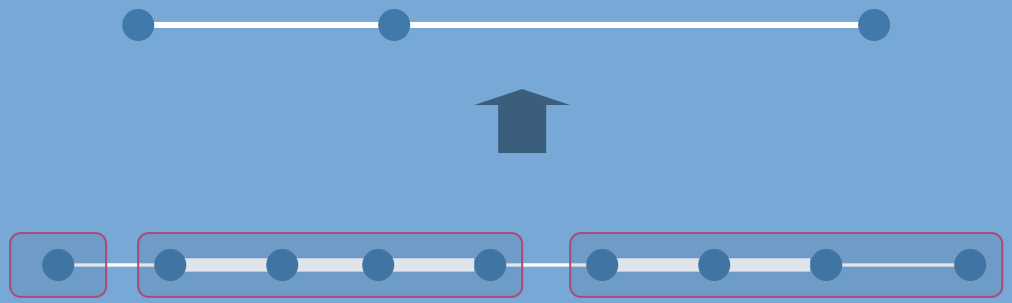
GAMG – Smoothed Aggregation

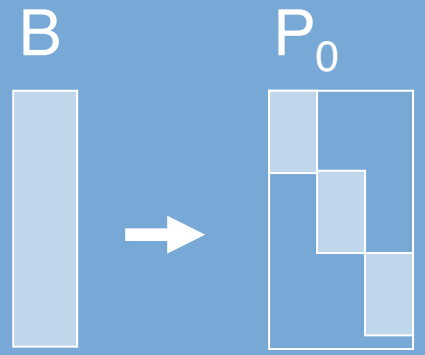
Piecewise constant function: “Plain” aggregation

Start with kernel vectors B of operator

eg, 6 rigid body modes in elasticity

Nodal aggregation





“Smoothed” aggregation: lower energy of functions

One Jacobi iteration: $P \leftarrow (I - \omega D^{-1} A) P_0$

BERKELEY LAB
Lawrence Berkeley National Laboratory

Outline

- High level discussion of multilevel iterative solver support in PETSc
- Algebraic multigrid (AMG) introduction
- PETSc's native AMG framework – GAMG
- Performance debugging GAMG

GAMG AMG framework in PETSc

- GAMG is designed to be modular, extensible, accept small contributions from users as easily as possible.
- GAMG base class drives generic AMG grid construction process (fine to coarse grids)
  - Galerkin coarse grid construction
  - Reduce number of active processors on coarse grid
  - Repartition coarse grids to keep good data locality.

Core AMG method construction

- P is what matters for 99% of AMG methods.
- P construction is decomposed into 4 methods (pure virtual functions)
  1. Strength of connection graph construction
     - Make symmetric scalar matrix (graph) w/ operator
  2. Coarsen: aggregate or select coarse grid points
  3. Create initial/tentative P – the AMG method
  4. Optimize P (optional), smoothing in SA

Instantiations/contributions to GAMG

- Graph construction (1)
  - – Symmetrize if needed $G = A + A^T$
  1. Create scalar from systems (eg, elasticity)
     - Sum |u(I,j)|
- Coarsening (3)
  1. Maximal Independent set (MIS), simple greedy
  2. Heavy edge matching (HEM), experimental, incrementally fuses heavy edges
  3. Column DOF plug in by Toby Isaac for ice sheet problems, thin flat 2.5 D

Instantiations/contributions to GAMG

- Construction of P (3)
  1. Aggregation, simple, fully supported
  2. Classical (Peter Brune), reference implementation
  3. Unstructured geometric AMG, reference impl.
- Optimization (1)
  1. Energy minimization with relaxation
     - Smoothed aggregation

Outline

- High level discussion of multilevel iterative solver support in PETSc
- Algebraic multigrid (AMG) introduction
- PETSc's native AMG framework – GAMG
- Performance debugging GAMG

Sources of convergence problems

- High-frequency Helmholtz – very hard
- Anisotropic grids and material properties
- Nearly incompressible elasticity without pressure variable formulations
- Large high-frequency jumps in coefficients and sharp jumps in coefficients
  - Domain Decomposition methods specialized for this in PCBDDC
- Thin bodies
- Unstable discretizations
- Ill-posed problems
- Any problem where the discretization of the problem is not accurate (linear tetrahedra?).
- Curl/Curl and Grad/Div need special purpose method
  - Hypre and ML have some (only have interfaces to hypre's)

AMG parameters

- And AMG is hard to “control” – no free lunch
  - Algebraic makes it easy to interface with and amortize code development, and makes completely unstructured robust in some sense (Galerkin coarse grids are mathematically powerful – a true projection)
  - But there is a price, hard to get optimal performance
- Coarsening rate is common parameter to optimize
  - Too slow and coarse grids are expensive
  - Too fast and convergence rate suffers
  - Also feedback, nonlinear “ODE”
    - Coarsen rate affects coarse grid stencil size, which affects coarsening rate

Coarsening rates

- ML and hypre have their own parameters, GAMG:
  - *-pc gamg threshold <x>* 0 <= x <~ 0.08
  - *-pc gamg square graph <N>* where N is # of levels to square
- Debugging: run with *–info* and grep on “GAMG”
- Coarsening rate should be about 3x in each dimension, roughly
- Also watch number of non-zeros – it will go up on coarse grids – a lot in fully 3D problems, by as much as ~10x after several levels
- Also watch times in *–log_summary* data
  - Time spent in Galerking coarse grid construction (MatPtAP) will be large on 3D problems – to simplify should spend about same time in Galerkin as in the solve phase (KSPSolve), much less in 2D … performance debugging

Performance debugging

- In general use *–log_view* (*log_summary* is depreciated)
- See where the time is going
  - Setup: PCSetup = MatPtAP + mesh setup
    - “mesh setup” pars are shown or = PCSetup - MatPtAP
  - Actual solve: KSPSolve – Setup
- Pick hot spots, dig down, ask us, repete
- PETSc has a lot of great stuff, but PETSc is not magic, think first, lots of things could be made better, you can help, that is how I got started.

Asymmetric operators

- Theory gets wobbly, often works but even when it looks OK it can bite you
- Example, BISICLES ice sheet modeling code uses cell center finite different of 2D nonlinear elasticity
  - Asymmetric
  - We are having problems converging linear and nonlinear solvers on very hard problems
  - We think the problem comes for the discretization
  - I think a symmetric discretization would work much better
- GAMG requires *-pc gamg sym graph*
- Hypre is better for asymetric problems – algorithm does not assume symmetry as much as smoothed aggregation

Chebyshev smoothers

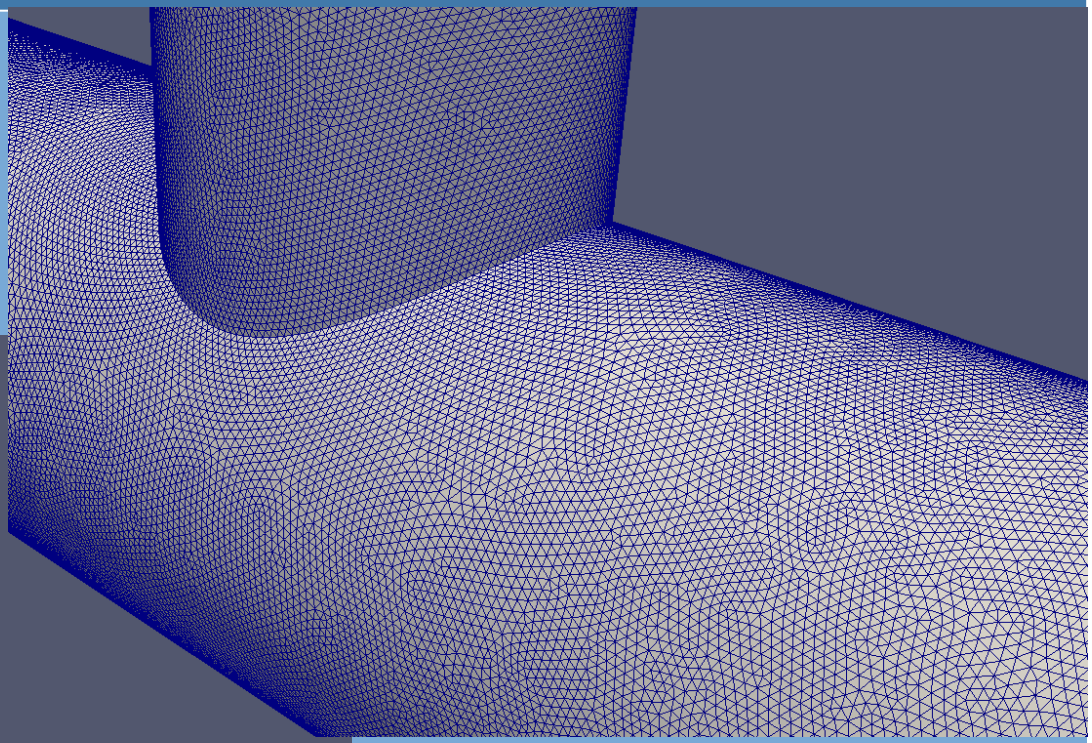
- Chebyshev smoothers are nice for AMG because Chebyshev is optimal for what you want: reduce error in a know part of the spectrum
- High end of spectrum is highest Eigen value – easy in principle to compute but not provable
  - This can bite you! A “bug” fix just made in GAMG.
  - If operator SPD use *-mg_levels_esteig_ksp_type_cg*
    - converges faster
  - If you get an “indefinite PC”, this could be the problem
- Low end of spectrum can be some ratio (eg, 30) of the highest – not a critical parameter

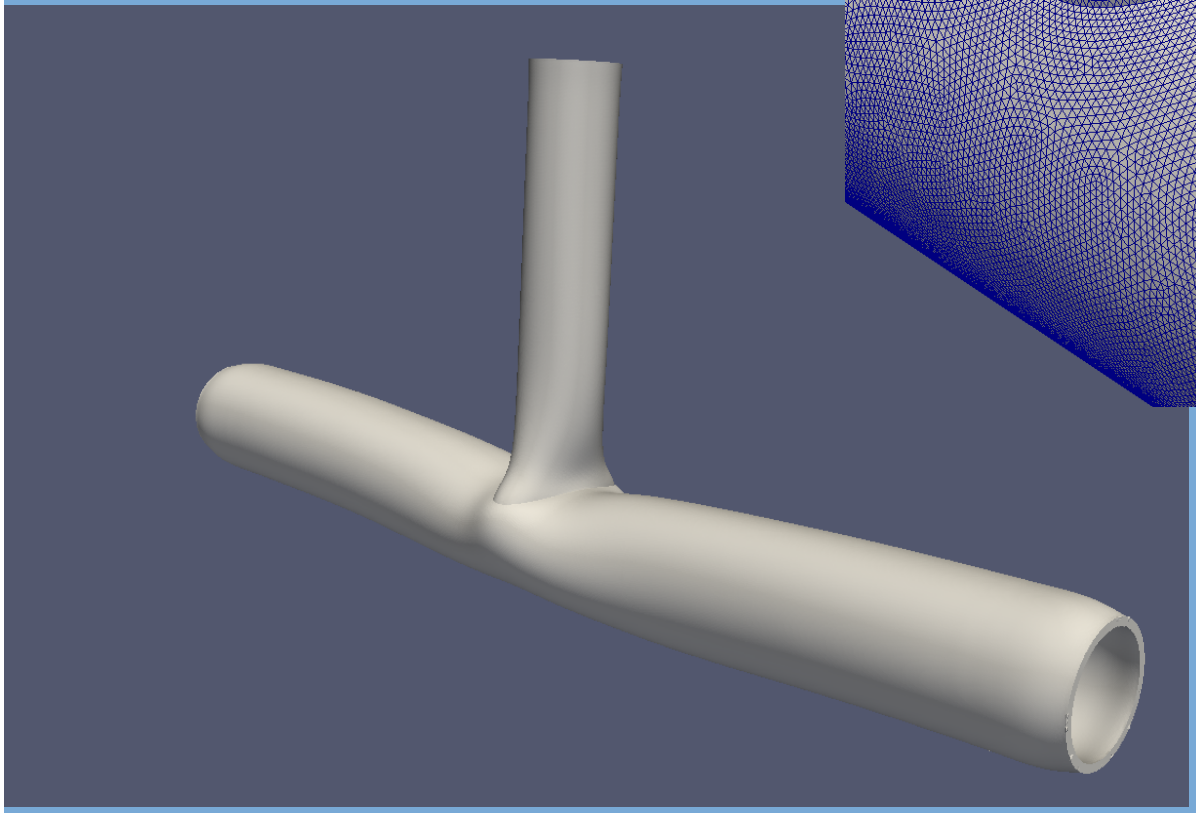
Multiple degrees-of-freedom

- If you have multiple dof/vertex or if you have a staggered grid, or other things, the kernel of your operator (without BCs) is not simply translation in each dof. Eg, rotational regid body modes, or zero energy modes, in elasticity.
- And you you should give GAMG the these kernel vectors.
- GAMG can take these kernels with MatSetNearNullSpace()
- PETSc provides utility for rigid body modes:
  - MatNullSpaceCreateRigidBody()
- You must also tell AMG solvers about this “block size” with *–mat_block_size bs*, and *–dm_block_size bs*
  - the solver may work without this but it is best to use it

Case study (work in progress) with Garth Wells….

Thank You





BERKELEY LAB
Lawrence Berkeley National Laboratory