# Overwhelmed with choices

- If you have a hard problem, no black-box solver will work well
- Everything in PETSc has a plugin architecture
  - Put in the “special sauce” for your problem
  - Your implementations are first-class
- PETSc exposes an algebra of composition at runtime
  - Build a good solver from existing components, at runtime
  - Multigrid, domain decomposition, factorization, relaxation, field-split
  - Choose matrix format that works best with your preconditioner
  - structural blocking, Neumann matrices, monolithic versus nested

# Questions to ask when you see a matrix

1. What do you want to do with it?
   - Multiply with a vector
   - Solve linear systems or eigen-problems
2. How is the conditioning/spectrum?
   - distinct/clustered eigen/singular values?
   - symmetric positive definite ($\sigma(A) \subset \mathbb{R}^+$)?
   - nonsymmetric definite ($\sigma(A) \subset \{z \in \mathbb{C} : \Re[z] > 0\}$)?
   - indefinite?
3. How dense is it?
   - block/banded diagonal?
   - sparse unstructured?
   - denser than we'd like?
4. Is there a better way to compute $Ax$?
5. Is there a different matrix with similar spectrum, but nicer properties?
6. How can we precondition $A$?

# Questions to ask when you see a matrix

1. What do you want to do with it?
   - Multiply with a vector
   - Solve linear systems or eigen-problems
2. How is the conditioning/spectrum?
   - distinct/clustered eigen/singular values?
   - symmetric positive definite ($\sigma(A) \subset \mathbb{R}^+$)?
   - nonsymmetric definite ($\sigma(A) \subset \{z \in \mathbb{C} : \Re[z] > 0\}$)?
   - indefinite?
3. How dense is it?
   - block/banded diagonal?
   - sparse unstructured?
   - denser than we'd like?
4. Is there a better way to compute $Ax$?
5. Is there a different matrix with similar spectrum, but nicer properties?
6. How can we precondition $A$?

# Preconditioning

**Definition (Preconditioner)**

A preconditioner $\mathcal{P}$ is a method for constructing a matrix $P^{-1} = \mathcal{P}(A, A_p)$ using a matrix $A$ and extra information $A_p$, such that the spectrum of $P^{-1}A$ (or $AP^{-1}$) is well-behaved.

- $P^{-1}$ is dense, $P$ is often not available and is not needed
- $A$ is rarely used by $\mathcal{P}$, but $A_p = A$ is common
- $A_p$ is often a sparse matrix, the "preconditioning matrix"
- Matrix-based: Jacobi, Gauss-Seidel, SOR, ILU(k), LU
- Parallel: Block-Jacobi, Schwarz, Multigrid, FETI-DP, BDDC
- Indefinite: Schur-complement, Domain Decomposition, Multigrid

# Preconditioning

## Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2 P^{-1}b, \dots\}$$

- Right preconditioning

$$(AP^{-1})Px = b$$

$$\{b, (P^{-1}A)b, (P^{-1}A)^2 b, \dots\}$$

- The product $P^{-1}A$ or $AP^{-1}$ is not formed.

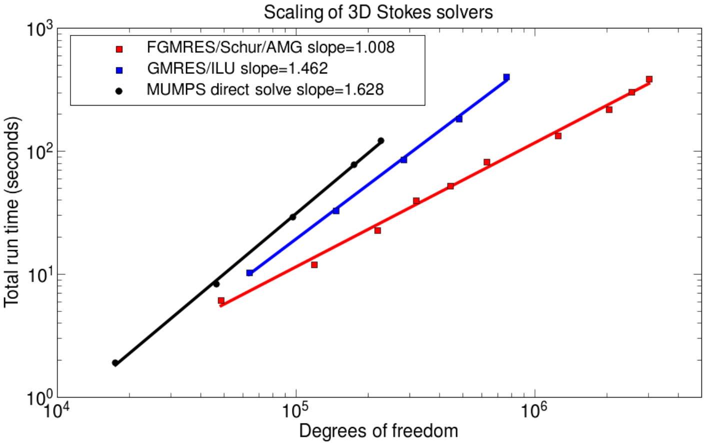
## Definition (Preconditioner)

A preconditioner $\mathcal{P}$ is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathcal{P}(A, A_p)$ using a matrix $A$ and extra information $A_p$, such that the spectrum of $P^{-1}A$ (or $AP^{-1}$) is well-behaved.

# Linear Solvers

- Use a direct method (small problem size)
- Precondition with Schur Complement method
- Use multigrid approach

# What about direct linear solvers?




- By all means, start with a direct solver
- Direct solvers are robust, but not scalable
- **2D**: $\mathcal{O}(n^{1.5})$ flops, $\mathcal{O}(n \log n)$ memory.
- **3D**: $\mathcal{O}(n^2)$ flops, $\mathcal{O}(n^{4/3})$ memory

# 3rd Party Solvers in PETSc

## Complete table of solvers

1. Sequential LU
   - ILUDT (SPARSEKIT2, Yousef Saad, U of MN)
   - EUCLID & PILUT (Hypre, David Hysom, LLNL)
   - ESSL (IBM)
   - SuperLU (Jim Demmel and Sherry Li, LBNL)
   - Matlab
   - UMFPACK (Tim Davis, U. of Florida)
   - LUSOL (MINOS, Michael Saunders, Stanford)
2. Parallel LU
   - MUMPS (Patrick Amestoy, IRIT)
   - SPOOLES (Cleve Ashcroft, Boeing)
   - SuperLU_Dist (Jim Demmel and Sherry Li, LBNL)
3. Parallel Cholesky
   - DSCPACK (Padma Raghavan, Penn. State)
4. XYTlib - parallel direct solver (Paul Fischer and Henry Tufo, ANL)

# 3rd Party Preconditioners in PETSc

## Complete table of solvers

1. Parallel ICC
   - BlockSolve95 (Mark Jones and Paul Plassman, ANL)
2. Parallel ILU
   - BlockSolve95 (Mark Jones and Paul Plassman, ANL)
3. Parallel Sparse Approximate Inverse
   - Parasails (Hypre, Edmund Chow, LLNL)
   - SPAI 3.0 (Marcus Grote and Barnard, NYU)
4. Sequential Algebraic Multigrid
   - RAMG (John Ruge and Klaus Steuben, GMD)
   - SAMG (Klaus Steuben, GMD)
5. Parallel Algebraic Multigrid
   - Prometheus (Mark Adams, PPPL)
   - BoomerAMG (Hypre, LLNL)
   - ML (Trilinos, Ray Tuminaro and Jonathan Hu, SNL)

# The Great Solver Schism: Monolithic or Split?

## Monolithic

- Direct solvers
- Coupled Schwarz
- Coupled Neumann-Neumann (need unassembled matrices)
- Coupled multigrid

X Need to understand local spectral and compatibility properties of the coupled system

## Split

- Physics-split Schwarz (based on relaxation)
- Physics-split Schur (based on factorization)
  - approximate commutators SIMPLE, PCD, LSC
  - segregated smoothers
  - Augmented Lagrangian
  - "parabolization" for stiff waves

X Need to understand global coupling strengths

- Preferred data structures depend on which method is used.
- Interplay with geometric multigrid.

# Outlook on Solver Composition

- Unintrusive composition of multigrid and block preconditioning
- We can build many preconditioners from the literature on the command line
- User code does not depend on matrix format, preconditioning method, nonlinear solution method, time integration method (implicit or IMEX), or size of coupled system (except for driver).

## In development

- Distributive relaxation, Vanka smoothers
- Algebraic coarsening of “dual” variables
- Improving operator-dependent semi-geometric multigrid
- More automatic spectral analysis and smoother optimization
- Automated support for mixing analysis into levels

# Stokes example

The common block preconditioners for Stokes require only options:

## The Stokes System

```
-pc_type fieldsplit
-pc_fieldsplit_type

-fieldsplit_0_ksp_type preonly
```

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}$$

# Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type additive

-fieldsplit_0_pc_type ml
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type jacobi
-fieldsplit_1_ksp_type preonly
```

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

Cohouet and Chabard, Some fast 3D finite element solvers for the generalized Stokes problem, 1988.

# Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type
multiplicative
-fieldsplit_0_pc_type hypre
-fieldsplit_0_ksp_type preonly
-fieldsplit_1_pc_type jacobi
-fieldsplit_1_ksp_type preonly
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

Elman, Multigrid and Krylov subspace methods for the discrete Stokes equations, 1994.

## Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres

-pc_fieldsplit_schur_factorization_type diag
```

$$\mathrm{PC} \quad \begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

Olshanskii, Peters, and Reusken, Uniform preconditioners for a parameter dependent saddle point problem with application to generalized Stokes interface equations, 2006.

# Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres

-pc_fieldsplit_schur_factorization_type lower
```

PC

$$\begin{pmatrix} \hat{A} & 0 \\ B^T & \hat{S} \end{pmatrix}$$

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

# Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres

-pc_fieldsplit_schur_factorization_type upper
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S} \end{pmatrix}$$

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

# Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type lsc
-fieldsplit_1_ksp_type minres

-pc_fieldsplit_schur_factorization_type upper
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S}_{LSC} \end{pmatrix}$$


May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

Kay, Loghin and Wathen, A Preconditioner for the Steady-State N-S Equations, 2002.
Elman, Howle, Shadid, Shuttleworth, and Tuminaro, Block preconditioners based on approximate commutators, 2006.

# Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
```

PC

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type
```

# System on each Coarse Level

$$R \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} P$$

# Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type additive

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type jacobi
-mg_levels_fieldsplit_1_ksp_type preonly
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

# Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type
multiplicative

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type jacobi
-mg_levels_fieldsplit_1_ksp_type preonly
```

Smoother PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

# Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type diag
```

# Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother PC

$$\begin{pmatrix} \hat{A} & 0 \\ B^T & \hat{S} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type lower
```

# Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother
PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type upper
```

# Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type lsc
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S}_{\mathrm{LSC}} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type upper
```

# Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Shur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

# Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Shur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

# Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Shur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

# Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Shur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

# Relative effect of the blocks

$$J = \begin{pmatrix} J_{uu} & J_{up} & J_{uE} \\ J_{pu} & 0 & 0 \\ J_{Eu} & J_{Ep} & J_{EE} \end{pmatrix}.$$

- $J_{uu}$ Viscous/momentum terms, nearly symmetric, variable coefficients, anisotropy from Newton.
- $J_{up}$ Weak pressure gradient, viscosity dependence on pressure (small), gravitational contribution (pressure-induced density variation). Large, nearly balanced by gravitational forcing.
- $J_{uE}$ Viscous dependence on energy, very nonlinear, not very large.
- $J_{pu}$ Divergence (mass conservation), nearly equal to $J_{up}^T$.
- $J_{Eu}$ Sensitivity of energy on momentum, mostly advective transport. Large in boundary layers with large thermal/moisture gradients.
- $J_{Ep}$ Thermal/moisture diffusion due to pressure-melting, $\boldsymbol{u} \cdot \nabla$.
- $J_{EE}$ Advection-diffusion for energy, very nonlinear at small regularization. Advection-dominated except in boundary layers

# How much nesting?

$$P_1 = \begin{pmatrix} J_{uu} & J_{up} & J_{uE} \\ 0 & B_{pp} & 0 \\ 0 & 0 & J_{EE} \end{pmatrix}$$

- $B_{pp}$ is a mass matrix in the pressure space weighted by inverse of kinematic viscosity.
- Elman, Mihajlović, Wathen, JCP 2011 for non-dimensional isoviscous Boussinesq.
- Works well for non-dimensional problems on the cube, not for realistic parameters.

$$P = \begin{bmatrix} \begin{pmatrix} J_{uu} & J_{up} \\ J_{pu} & 0 \end{pmatrix} & \\ \begin{pmatrix} J_{Eu} & J_{Ep} \end{pmatrix} & J_{EE} \end{bmatrix}$$

- Inexact inner solve using upper-triangular with $B_{pp}$ for Schur.
- Another level of nesting.
- GCR tolerant of inexact inner solves.
- Outer converges in 1 or 2 iterations.

- Low-order preconditioning full-accuracy unassembled high order operator.

# Why do we need multilevel solvers?

- Elliptic problems are globally coupled
- Without a coarse level, number of iterations proportional to inverse mesh size
- High-volume local communication is an inefficient way to communicate long-range information, bad for parallel models
- Most important with 3D flow features and/or slippery beds
- Nested/split multilevel methods
  - Decompose problem into simpler sub-problems, use multilevel methods on each
  - Good reuse of existing software
  - More synchronization due to nesting, more suitable after linearization
- Monolithic/coupled multilevel methods
  - Better convergence and lower synchronization, but harder to get right
  - Internal nonlinearities resolved locally
  - More discretization-specific, less software reuse

# Multigrid

Multigrid is optimal in that is does $\mathcal{O}(N)$ work for $||r|| < \epsilon$

- Brandt, Briggs, Chan & Smith
- Constant work per level
  - Sufficiently strong solver
  - Need a constant factor decrease in the residual
- Constant factor decrease in dof
  - Log number of levels

# Multilevel Solvers are a Way of Life

- ingredients that discretizations can provide
  - identify “fields”
  - topological coarsening, possibly for fields
  - near-null space information
  - “natural” subdomains
  - subdomain integration, face integration
  - element or subdomain assembly/matrix-free smoothing
- solver composition
  - most splitting methods accessible from command line
  - energy optimization for tentative coarse basis functions
  - algebraic form of distributive relaxation
  - generic assembly for large systems and components
  - working on flexibile “library-assisted” nonlinear multigrid
  - adding support for interactive eigenanalysis

# Linear Multigrid

Smoothing (typically Gauss-Seidel)

$$x^{new} = S(x^{old}, b) \tag{1}$$

Coarse-grid Correction

$$\begin{aligned} J_c \delta x_c &= R(b - Jx^{old}) && (2) \\ x^{new} &= x^{old} + R^T \delta x_c && (3) \end{aligned}$$

# Multigrid

### Hierarchy: Interpolation and restriction operators

$$\mathcal{I}^{\uparrow} : X_{\text{coarse}} \to X_{\text{fine}} \qquad \mathcal{I}^{\downarrow} : X_{\text{fine}} \to X_{\text{coarse}}$$

- Geometric: define problem on multiple levels, use grid to compute hierarchy
- Algebraic: define problem only on finest level, use matrix structure to build hierarchy

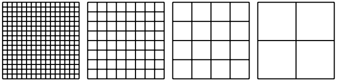
### Galerkin approximation

Assemble this matrix: $A_{\text{coarse}} = \mathcal{I}^{\downarrow} A_{\text{fine}} \mathcal{I}^{\uparrow}$

### Application of multigrid preconditioner ($V$-cycle)

- Apply pre-smoother on fine level (any preconditioner)
- Restrict residual to coarse level with $\mathcal{I}^{\downarrow}$
- Solve on coarse level $A_{\text{coarse}} x = r$
- Interpolate result back to fine level with $\mathcal{I}^{\uparrow}$
- Apply post-smoother on fine level (any preconditioner)

# Multigrid Preliminaries



**Multigrid** is an $O(n)$ method for solving algebraic problems by defining a hierarchy of scale. A multigrid method is constructed from:

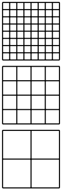
1. a series of discretizations
   - coarser approximations of the original problem
   - constructed algebraically or geometrically
2. intergrid transfer operators
   - residual restriction $I_h^H$ (fine to coarse)
   - state restriction $\hat{I}_h^H$ (fine to coarse)
   - partial state interpolation $I_H^h$ (coarse to fine, 'prolongation')
   - state reconstruction $\mathbb{I}_H^h$ (coarse to fine)
3. Smoothers ($S$)
   - correct the high frequency error components
   - Richardson, Jacobi, Gauss-Seidel, etc.
   - Gauss-Seidel-Newton or optimization methods

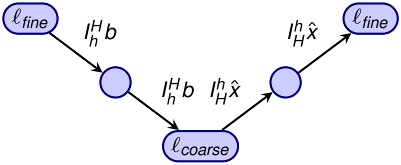
# Rediscretized Multigrid using `DM`

- `DM` manages problem data beyond purely algebraic objects
  - structured, redundant, and (less mature) unstructured implementations in PETSc
  - third-party implementations
- `DMCoarsen(dmfine,coarse_comm,&coarsedm)` to create "geometric" coarse level
  - Also `DMRefine()` for grid sequencing and convenience
  - `DMCoarsenHookAdd()` for external clients to move resolution-dependent data for rediscretization and FAS
- `DMCreateInterpolation(dmcoarse,dmfine,&Interp,&Rscale)`
  - Usually uses geometric information, can be operator-dependent
  - Can be improved subsequently, e.g. using energy-minimization from AMG
- Resolution-dependent solver-specific callbacks use attribute caching on `DM`.
  - Managed by solvers, not visible to users unless they need exotic things (e.g. custom homogenization, reduced models)

# Multigrid

- **Multigrid** methods uses coarse correction for large-scale error

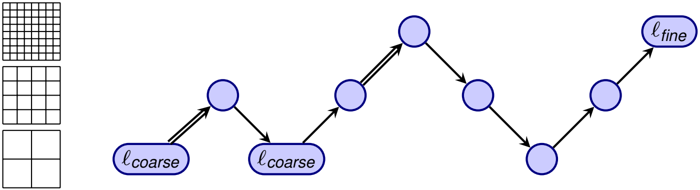





Algorithm $MG(A, b)$ for the solution of $A\vec{x} = b$:

$$\begin{aligned}
\vec{x} &= S^m(\vec{x}, b) && \text{pre-smooth} \\
b^H &= I_h^H(\vec{r} - A\vec{x}) && \text{restrict residual} \\
\hat{x}^H &= MG(I_h^H A I_H^h, b^H) && \text{recurse} \\
\vec{x} &= \vec{x} + I_H^h \hat{x}^H && \text{prolong correction} \\
\vec{x} &= \vec{x} + S^n(\vec{x}, b) && \text{post-smooth}
\end{aligned}$$

# Full Multigrid(FMG)




- start wich coarse grid
- $\vec{x}$ is prolonged using $\mathbb{I}_H^h$ on first visit to each finer level
- truncation error within one cycle
- about five work units for many problems
- highly efficient solution method

# Some Multigrid Options

- `-snes_grid_sequence`: [0]
  Solve nonlinear problems on coarse grids to get initial guess
- `-pc_mg_galerkin`: [FALSE]
  Use Galerkin process to compute coarser operators
- `-pc_mg_type`: [FULL]
  (choose one of) MULTIPLICATIVE ADDITIVE FULL KASKADE
- `-mg_coarse_{ksp,pc}_*`
  control the coarse-level solver
- `-mg_levels_{ksp,pc}_*`
  control the smoothers on levels
- `-mg_levels_3_{ksp,pc}_*`
  control the smoother on specific level
- These also work with ML's algebraic multigrid.

# Coupled Multigrids

- Geometric multigrid with isotropic coarsening, ASM(1)/Cholesky and ASM(0)/ICC(0) on levels

  `-mg_levels_pc_type bjacobi -mg_levels_sub_pc_type icc -mg_levels_1_pc_type asm -mg_levels_1_sub_pc_type cholesky`

- ...with Galerkin coarse operators

  `-pc_mg_galerkin`

- ...with ML's aggregates

  `-pc_type ml -mg_levels_pc_type asm`

- Geometric multigrid with aggressive semi-coarsening, ASM(1)/Cholesky and ASM(0)/ICC(0) on levels

  `-da_refine_hierarchy_x 1,1,8,8 -da_refine_hierarchy_y 2,2,1,1 -da_refine_hierararchy_z 2,2,1,1`

- Simulate 1024 cores, interactively, on my laptop

  `-mg_levels_pc_asm_blocks 1024`

# Everything is better as a smoother (sometimes)

## Block preconditioners work alright, but...

- nested iteration requires more dot products
- more iterations: coarse levels don't "see" each other
- finer grained kernels: lower arithmetic intensity, even more limited by memory bandwidth

## Coupled multigrid

- need compatible coarsening
  - can do algebraically (Adams 2004) but would need to assemble
- stability issues for lowest order $Q_1 - P_0^{\text{disc}}$
  - Rannacher-Turek looks great, but no discrete Korn's inequality
- coupled "Vanka" smoothers difficult to implement with high performance, especially for FEM
- block preconditioners as smoothers reuse software better
- one level by reducing order for the coarse space, more levels need non-nested geometric MG or go all-algebraic and pay for matrix assembly and setup

# Multigrid convergence properties

- Textbook: $P^{-1}A$ is spectrally equivalent to identity
  - Constant number of iterations to converge up to discretization error
- Most theory applies to SPD systems
  - variable coefficients (e.g. discontinuous): low energy interpolants
  - mesh- and/or physics-induced anisotropy: semi-coarsening/line smoothers
  - complex geometry: difficult to have meaningful coarse levels
- Deeper algorithmic difficulties
  - nonsymmetric (e.g. advection, shallow water, Euler)
  - indefinite (e.g. incompressible flow, Helmholtz)
- Performance considerations
  - Aggressive coarsening is critical in parallel
  - Most theory uses SOR smoothers, ILU often more robust
  - Coarsest level usually solved semi-redundantly with direct solver
- Multilevel Schwarz is essentially the same with different language
  - assume strong smoothers, emphasize aggressive coarsening

# Algebraic Multigrid Tuning

- Smoothed Aggregation (GAMG, ML)
  - Graph/strength of connection – MatSetBlockSize()
  - Threshold (`-pc_gamg_threshold`)
  - Aggregate (MIS, HEM)
  - Tentative prolongation – MatSetNearNullSpace()
  - Eigenvalue estimate
  - Chebyshev smoothing bounds
- BoomerAMG (Hypre)
  - Strong threshold (`-pc_hypre_boomeramg_strong_threshold`)
  - Aggressive coarsening options

# Coupled approach to multiphysics

- Smooth all components together
  - Block SOR is the most popular
  - Block ILU sometimes more robust (e.g. transport/anisotropy)
  - Vanka field-split smoothers or for saddle-point problems
  - Distributive relaxation
- Scaling between fields is critical
- Indefiniteness
  - Make smoothers and interpolants respect inf-sup condition
  - Difficult to handle anisotropy
  - Exotic interpolants for Helmholtz
- Transport
  - Define smoother in terms of first-order upwind discretization ($h$-ellipticity)
  - Evaluate residuals using high-order discretization
  - Use Schur field-split: "parabolize" at top level or for smoother on levels
- Multigrid inside field-split or field-split inside multigrid
- Open research area, hard to write modular software

ex55: Allen-Cahn problem in 2D

- constant mobility
- triangular elements

Geometric multigrid method for saddle point variational inequalities:

```
./ex55 -ksp_type fgmres -pc_type mg -mg_levels_ksp_type fgmres
-mg_levels_pc_type fieldsplit -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_pc_fieldsplit_type schur -da_grid_x 65 -da_grid_y 65
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition user
-mg_levels_fieldsplit_1_ksp_type gmres -mg_coarse_ksp_type preonly
-mg_levels_fieldsplit_1_pc_type none   -mg_coarse_pc_type svd
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor    -pc_mg_levels 5
-mg_levels_fieldsplit_0_pc_sor_forward -pc_mg_galerkin
-snes_vi_monitor -ksp_monitor_true_residual -snes_atol 1.e-11
-mg_levels_ksp_monitor -mg_levels_fieldsplit_ksp_monitor
-mg_levels_ksp_max_it 2 -mg_levels_fieldsplit_ksp_max_it 5
```