Getting Modern Algorithms into the Hands of Working Scientists on Modern Hardware

Matthew Knepley

Computation Institute
University of Chicago

Department of Molecular Biology and Physiology
Rush University Medical Center

Bridging the Gap Between the
Geosciences and Mathematics, Statistics, and Computer Science
Princeton, NJ October 1–2, 2012

Impact of this Project

The main impact of mathematics will be

design/analysis of algorithms
for simulation & data analysis

This is where CS comes in . . .

Impact of this Project

The main impact of mathematics will be

design/analysis of algorithms

for simulation & data analysis

This is where CS comes in . . .

The main impact of mathematics will be

design/analysis of algorithms for simulation & data analysis

This is where CS comes in . . .

Impact of this Project

The main impact of mathematics will be

design/analysis of algorithms for simulation & data analysis

This is where CS comes in . . .

Outline

1. Computational Science
   - Linear Algebra
   - FEM Integration

2. Mathematics

Big Idea

The best way to create robust,

efficient and scalable,

maintainable scientific codes,

is to use libraries.

Big Idea

The best way to create robust, efficient and scalable, maintainable scientific codes,

is to use libraries.

Big Idea

The best way to create robust, efficient and scalable, maintainable scientific codes,

is to use libraries.

Big Idea

The best way to create robust, efficient and scalable, maintainable scientific codes,

is to use libraries.

Why Libraries?

- Hides hardware details

  MPI does for this for machines and networks

- Hide Implementation Complexity

  PETSc does for this Matrices and Krylov Solvers

Why Libraries?

- Hides hardware details

  MPI does for this for machines and networks

- Hide Implementation Complexity

  PETSc does for this Matrices and Krylov Solvers

Why GPUs?

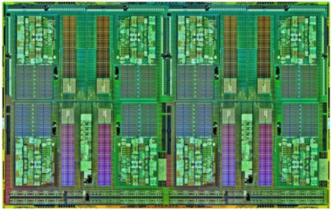
In the next 10 years, every machine will

Why GPUs?

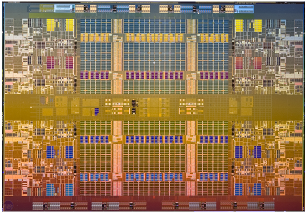
In the next 10 years, every machine will at least have multicores, 2–16 cores,

Why GPUs?

In the next 10 years, every machine will at least have multicores, 2–16 cores,



AMD Interlagos



Intel Nehalem Beckton

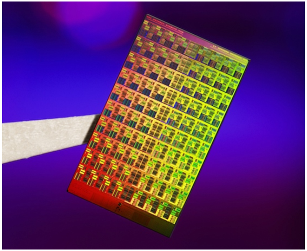
Why GPUs?

In the next 10 years, every machine will probably have manycores, 100–1000 cores.

Why GPUs?

In the next 10 years, every machine will probably have manycores, 100–1000 cores.

NVidia C2070



Intel MIC

Outline

1. Computational Science
   - Linear Algebra
   - FEM Integration

VECCUDA

Strategy: Define a new **Vec** implementation

- Uses Thrust for data storage and operations on GPU
- Supports full PETSc **Vec** interface
- Inherits PETSc scalar type
- Can be activated at runtime, `-vec_type cuda`
- PETSc provides memory coherence mechanism

MATAIJCUDA

Also define new **Mat** implementations

- Uses Cusp for data storage and operations on GPU
- Supports full PETSc **Mat** interface, some ops on CPU
- Can be activated at runtime, `-mat_type aijcuda`
- Notice that parallel matvec necessitates off-GPU data transfer

Solvers

Solvers come for Free

Preliminary Implementation of PETSc Using GPU,

Minden, Smith, Knepley, 2010

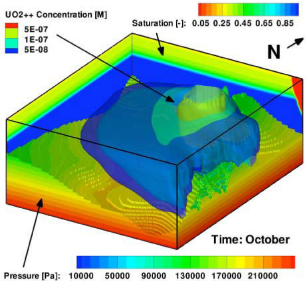
- All linear algebra types work with solvers
- Entire solve can take place on the GPU
  - Only communicate scalars back to CPU
- GPU communication cost could be amortized over several solves
- Preconditioners are a problem
  - Cusp has a promising AMG

Example
PFLOTRAN

Flow Solver

$32 \times 32 \times 32$ grid

Routine	Time (s)	MFlops	MFlops/s
CPU			
KSPSolve	8.3167	4370	526
MatMult	1.5031	769	512
GPU			
KSPSolve	1.6382	4500	2745
MatMult	0.3554	830	2337



P. Lichtner, G. Hammond, R. Mills, B. Phillip

Example

Driven Cavity Velocity-Vorticity with Multigrid

```
ex50 -da_vec_type seqcusp
  -da_mat_type aijcusp -mat_no_inode # Setup types
  -da_grid_x 100 -da_grid_y 100      # Set grid size
  -pc_type none -pc_mg_levels 1      # Setup solver
  -preload off -cuda_synchronize     # Setup run
  -log_summary
```

Outline

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

Interface Maturity

Some parts of PDE computation are less mature

Linear Algebra

- One universal interface
  - BLAS, PETSc, Trilinos, FLAME, Elemental
- Entire problem can be phrased in the interface
  - $Ax = b$
- Standalone component

Finite Elements

- Many Interfaces
  - FEniCS, FreeFEM++, DUNE, dealII, Fluent
- Problem definition requires general code
  - Physics, boundary conditions
- Crucial interaction with other simulation components
  - Discretization, mesh/geometry

FEM Integration Model

Proposed by Jed Brown

We consider weak forms dependent only on fields and gradients,

$$\int_\Omega \phi \cdot f_0(u, \nabla u) + \nabla\phi : \vec{f}_1(u, \nabla u) = 0. \quad (1)$$

Discretizing we have

$$\sum_e \mathcal{E}_e^T \left[B^T W^q f_0(u^q, \nabla u^q) + \sum_k D_k^T W^q \vec{f}_1^k(u^q, \nabla u^q) \right] = 0 \quad (2)$$

f_n	pointwise physics functions
u^q	field at a quad point
W^q	diagonal matrix of quad weights
B,D	basis function matrices which reduce over quad points
$\mathcal{E}$	assembly operator

PETSc Integration

PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		Automation		Solver Input
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

- User provides point-wise physics functions
- Loops are done in batches, remainder cells handled by CPU
- One batch integration method with compile-time sizes
  - CPU, multicore CPU, MIC, GPU, etc.
- PETSc ex52 is a single-field example

PETSc Integration

PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		Automation		Solver Input
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

- User provides point-wise physics functions
- Loops are done in batches, remainder cells handled by CPU
- One batch integration method with compile-time sizes
  - CPU, multicore CPU, MIC, GPU, etc.
- PETSc ex52 is a single-field example

PETSc Integration

PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		Automation		Solver Input
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

- User provides point-wise physics functions
- Loops are done in batches, remainder cells handled by CPU
- One batch integration method with compile-time sizes
  - CPU, multicore CPU, MIC, GPU, etc.
- PETSc ex52 is a single-field example

PETSc Integration

PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		Automation		Solver Input
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

- User provides point-wise physics functions
- Loops are done in batches, remainder cells handled by CPU
- One batch integration method with compile-time sizes
  - CPU, multicore CPU, MIC, GPU, etc.
- PETSc ex52 is a single-field example

PETSc Integration

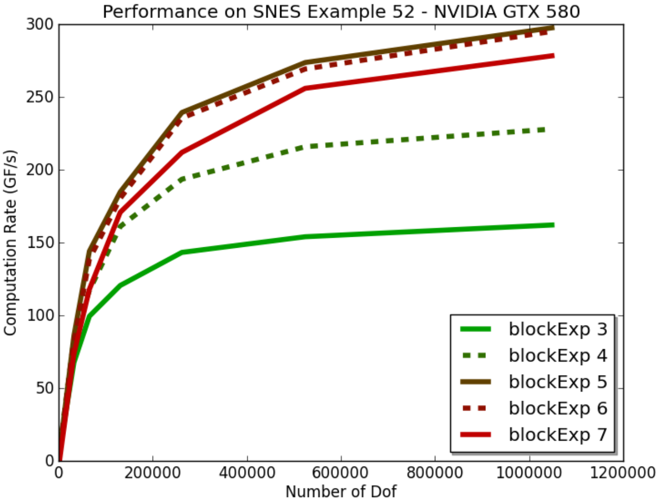
PETSc FEM Organization

GPU evaluation is transparent to the user:

User Input		Automation		Solver Input
domain	==	Triangle/TetGen	==>	Mesh
element	==	FIAT	==>	Tabulation
f_n	==	Generic Evaluation	==>	Residual

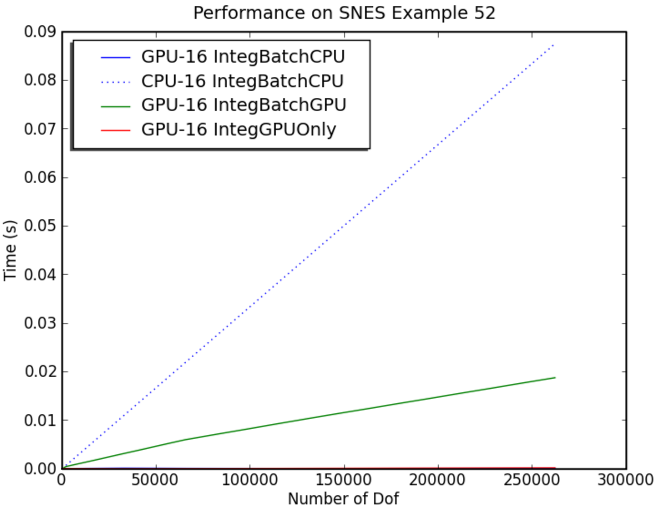
- User provides point-wise physics functions
- Loops are done in batches, remainder cells handled by CPU
- One batch integration method with compile-time sizes
  - CPU, multicore CPU, MIC, GPU, etc.
- PETSc ex52 is a single-field example

2D P_1 Laplacian Performance



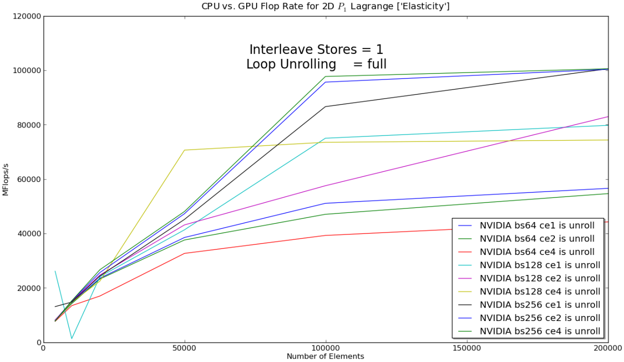
Reaches 100 GF/s by 100K elements

2D P_1 Laplacian Performance



Linear scaling for both GPU and CPU integration

2D P_1 Rate-of-Strain Performance



Reaches 100 GF/s by 100K elements

General Strategy

- Vectorize
- Overdecompose
- Cover memory latency with computation
  - Multiple cycles of writes in the kernel
- User must relinquish control of the layout

Finite Element Integration on GPUs, accepted ACM TOMS,
Andy Terrel and Matthew Knepley.
Finite Element Integration with Quadrature on the GPU, to SISC,
Robert Kirby, Matthew Knepley, Andreas Klöckner, and Andy Terrel.

Outline

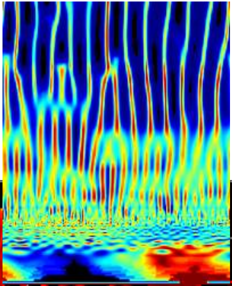
Composable System for Scalable Preconditioners

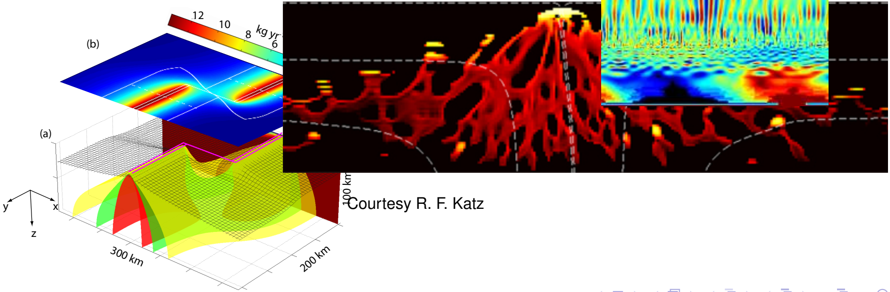
Stokes and KKT

The saddle-point matrix is a canonical form for handling constraints:

- Incompressibility
- Contact
- Multi-constituent phase-field models
- Optimal control
- PDE constrained optimization

Courtesy M. Spiegelman





Courtesy R. F. Katz

Composable System for Scalable Preconditioners

Stokes and KKT

There are *many* approaches for saddle-point problems:

- Block preconditioners
- Schur complement methods
- Multigrid with special smoothers

$$\begin{pmatrix} F & B & M \\ B^T & 0 & 0 \\ N & 0 & K \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \\ T \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \\ q \end{pmatrix}$$

However, today it is hard to compare & combine them and combine in a **hierarchical** manner. For instance we might want,

Composable System for Scalable Preconditioners

Stokes and KKT

There are *many* approaches for saddle-point problems:

- Block preconditioners
- Schur complement methods
- Multigrid with special smoothers

$$\begin{pmatrix} F & B & M \\ B^T & 0 & 0 \\ N & 0 & K \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \\ T \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \\ q \end{pmatrix}$$

However, today it is hard to compare & combine them and combine in a **hierarchical** manner. For instance we might want,

Composable System for Scalable Preconditioners
Stokes and KKT

There are *many* approaches for saddle-point problems:

- Block preconditioners
- Schur complement methods
- Multigrid with special smoothers

$$\begin{pmatrix} F & B & M \\ B^T & 0 & 0 \\ N & 0 & K \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \\ T \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \\ q \end{pmatrix}$$

However, today it is hard to compare & combine them and combine in a **hierarchical** manner. For instance we might want,

a Gauss-Siedel iteration between blocks of $(\mathbf{u}, p)$ and T,
and a full Schur complement factorization for **u** and p.

Composable System for Scalable Preconditioners

Stokes and KKT

There are *many* approaches for saddle-point problems:

- Block preconditioners
- Schur complement methods
- Multigrid with special smoothers

$$\begin{pmatrix} F & B & M \\ B^T & 0 & 0 \\ N & 0 & K \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \\ T \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \\ q \end{pmatrix}$$

However, today it is hard to compare & combine them and combine in a **hierarchical** manner. For instance we might want,

> an upper triangular Schur complement factorization for $\mathbf{u}$ and p, and geometric multigrid for the $\mathbf{u}$ block.

Composable System for Scalable Preconditioners
Stokes and KKT

There are *many* approaches for saddle-point problems:

- Block preconditioners
- Schur complement methods
- Multigrid with special smoothers

$$\begin{pmatrix} F & B & M \\ B^T & 0 & 0 \\ N & 0 & K \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \\ T \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \\ q \end{pmatrix}$$

However, today it is hard to compare & combine them and combine in a **hierarchical** manner. For instance we might want,

algebraic multigrid for the full $(\mathbf{u}, p)$ system,
using a block triangular Gauss-Siedel smoother on each level,
and use identity for the (p, p) block.

Approach for efficient, robust, scalable linear solvers

Need solvers to be:

- **Composable**: separately developed solvers may be easily combined, by non-experts, to form a more powerful solver
- **Nested**: outer solvers call inner solvers
- **Hierarchical**: outer solvers may iterate over all variables for a global problem, while nested inner solvers handle smaller subsets of physics, smaller physical subdomains, or coarser meshes
- **Extensible**: users can easily customize/extend

Composable Linear Solvers for Multiphysics, IPDPS, 2012,
J. Brown, M. G. Knepley, D. A. May, L. C. McInnes and B. F. Smith.

Stokes example

The common block preconditioners for Stokes require only options:

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

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_field_split_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 & 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_field_split_type multiplic

-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_field_split_type schur
-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly
-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
```

PC

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

```
-pc_fieldsplit_schur_factorization_type diag
```

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_field_split_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
```

PC

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

```
-pc_fieldsplit_schur_factorization_type lower
```

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_field_split_type schur
-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly
-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
```

PC

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

```
-pc_fieldsplit_schur_factorization_type upper
```

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_field_split_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type lsc
-fieldsplit_1_ksp_type minres
```

PC

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

```
-pc_fieldsplit_schur_factorization_type upper
```

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_field_split_type schur
-pc_fieldsplit_schur_factorization_type full
```

PC

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

Programming with Options

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
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

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
```

Schur 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
```

Schur 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
```

Schur 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
```

Schur 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
```

Composability

Composable interfaces allow the nested, hierarchical interaction of different components,

- **analysis** (discretization)
- **topology** (mesh)
- **algebra** (solver)

so that non-experts can produce powerful simulations with modern algorithms.

Jed Brown will discuss this interplay
in the context of multilevel solvers

Composability

Composable interfaces allow the nested, hierarchical interaction of different components,

- **analysis** (discretization)
- **topology** (mesh)
- **algebra** (solver)

so that non-experts can produce powerful simulations with modern algorithms.

Jed Brown will discuss this interplay in the context of multilevel solvers

Composability

Composable interfaces allow the nested, hierarchical interaction of different components,

- **analysis** (discretization)
- **topology** (mesh)
- **algebra** (solver)

so that non-experts can produce powerful simulations with modern algorithms.

Jed Brown will discuss this interplay
in the context of multilevel solvers

Composability

Composable interfaces allow the nested, hierarchical interaction of different components,

- **analysis** (discretization)
- **topology** (mesh)
- **algebra** (solver)

so that non-experts can produce powerful simulations with modern algorithms.

Jed Brown will discuss this interplay
in the context of multilevel solvers

Main Points

- Libraries encapsulate the Mathematics
  - Users will give up more Control
- Multiphysics demands Composable Solvers
  - Each piece will have to be Optimal

Main Points

- Libraries encapsulate the Mathematics
  - Users will give up more Control
- Multiphysics demands Composable Solvers
  - Each piece will have to be Optimal

Main Points

- Libraries encapsulate the Mathematics
  - Users will give up more Control
- Multiphysics demands Composable Solvers
  - Each piece will have to be Optimal

Main Points

- Libraries encapsulate the Mathematics
  - Users will give up more Control
- Multiphysics demands Composable Solvers
  - Each piece will have to be Optimal

Change alone is unchanging

— Heraclitus, 544–483 BC