The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley\(^1\) and PETSc Team

\(^1\)Computation Institute
University of Chicago

39th SPEEDUP Workshop on High-Performance Computing
ETH Zurich
Switzerland, September 6–7, 2010
Outline

1. What can PETSc do?
   - What is PETSc?
   - Who uses PETSc?

2. What’s New in PETSc?

3. Conclusion
Outline

1. What can PETSc do?
   - What is PETSc?
   - Who uses PETSc?
How did PETSc Originate?

PETSc was developed as a Platform for Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms
  - which blur these boundaries
Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

— Barry Smith
You want to think about how you decompose your data structures, how you think about them globally. [...] If you were building a house, you’d start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn’t start with a bunch of tiles and say, “Well I’ll put this tile down on the ground, and then I’ll find a tile to go next to it.” But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you’re going to survive making your code parallel.

(http://www.rce-cast.com/Podcast/rce-28-mpich2.html)
What is PETSc?

A freely available and supported research code
- Free for everyone, including industrial users
- Hyperlinked manual, examples, and manual pages for all routines
- Hundreds of tutorial-style examples
- Support via email: petsc-maint@mcs.anl.gov
- Usable from C, C++, Fortran 77/90, and Python
What can PETSc do?

Portable to any parallel system supporting MPI, including:
- Tightly coupled systems
  - Cray XT5, BG/Q, NVIDIA Fermi, Earth Simulator
- Loosely coupled systems, such as networks of workstations
  - IBM, Mac, iPad/iPhone, PCs running Linux or Windows

PETSc History
- Begun September 1991
- Over 60,000 downloads since 1995 (version 2)
- Currently 400 per month

PETSc Funding and Support
- Department of Energy
  - SciDAC, MICS Program, AMR Program, INL Reactor Program
- National Science Foundation
  - CIG, CISE, Multidisciplinary Challenge Program
What can PETSc do?

What is PETSc?

The PETSc Team

Bill Gropp

Barry Smith

Satish Balay

Jed Brown

Matt Knepley

Lisandro Dalcin

Hong Zhang

Victor Eijkhout

Dmitry Karpeev
Outline

1. What can PETSc do?
   - What is PETSc?
   - Who uses PETSc?
Computational Scientists

Earth Science
- PyLith (CIG)
- Underworld (Monash)
- Magma Dynamics (LDEO, Columbia)

Subsurface Flow and Porous Media
- STOMP (DOE)
- PFLOTRAN (DOE)
Who Uses PETSc?

Computational Scientists

- CFD
  - Fluidity
  - OpenFOAM
  - freeCFD
  - OpenFVM

- MicroMagnetics
  - MagPar

- Fusion
  - NIMROD
Algorithm Developers

- **Iterative methods**
  - Deflated GMRES
  - LGMRES
  - QCG
  - SpecEst

- **Preconditioning researchers**
  - Prometheus (Adams)
  - ParPre (Eijkhout)
  - FETI-DP (Klawonn and Rheinbach)
Who Uses PETSc?

Algorithm Developers

- Finite Elements
  - PETSc-FEM
  - libMesh
  - Deal II
  - OOFEM

- Other Solvers
  - Fast Multipole Method (PetFMM)
  - Radial Basis Function Interpolation (PetRBF)
  - Eigensolvers (SLEPc)
  - Optimization (TAO)
PETSc has run implicit problems with over 500 billion unknowns
- UNIC on BG/P and XT5
- PFLOTRAN for flow in porous media

PETSc has run on over 290,000 cores efficiently
- UNIC on the IBM BG/P Intrepid at ANL
- PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 22 Teraflops
- Kaushik on XT5
- LANL PFLOTRAN code
PETSc has run implicit problems with over 500 billion unknowns
- UNIC on BG/P and XT5
- PFLOTRAN for flow in porous media

PETSc has run on over 290,000 cores efficiently
- UNIC on the IBM BG/P Intrepid at ANL
- PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 22 Teraflops
- Kaushik on XT5
- LANL PFLOTRAN code
PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media

PETSc has run on over 290,000 cores efficiently
  - UNIC on the IBM BG/P Intrepid at ANL
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 22 Teraflops
  - Kaushik on XT5
  - LANL PFLOTRAN code
PyLith

- Multiple problems
  - Dynamic rupture
  - Quasi-static relaxation
- Multiple models
  - Nonlinear visco-plastic
  - Finite deformation
  - Fault constitutive models
- Multiple meshes
  - 1D, 2D, 3D
  - Hex and tet meshes
- Parallel
  - PETSc solvers
  - Sieve mesh management

\[a\] Aagaard, Knepley, Williams
Multiple Mesh Types

- Triangular
- Tetrahedral
- Rectangular
- Hexahedral
Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

---

Katz, Speigelman
Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

\(^{a}\text{Katz, Speigelman}\)
Fracture Mechanics

- Full variational formulation
  - Phase field
  - Linear or Quadratic penalty

- Uses TAO optimization
  - Necessary for linear penalty
  - Backtacking

- No prescribed cracks
  - Arbitrary crack geometry
  - Arbitrary intersections

- Multiple materials
  - Composite toughness

---

\(^a^\text{Bourdin}\)
Fracture Mechanics

1 Bourdin

M. Knepley ( )

PETSc 9/6/10 18 / 67
Vortex Method

\(t = 000\)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley} \)
**What can PETSc do?**

- **Incompressible Flow**
  - Gaussian vortex blobs
  - High Re

- **PetFMM**
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- **PetRBF**
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- **Parallelism**
  - MPI
  - GPU

---

**Who uses PETSc?**

Cruz, Yokota, Barba, Knepley

\[ ^a \text{Cruz, Yokota, Barba, Knepley} \]
Vortex Method

\[ t = 200 \]

- **Incompressible Flow**
  - Gaussian vortex blobs
  - High Re

- **PetFMM**
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- **PetRBF**
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- **Parallelism**
  - MPI
  - GPU

\[^{a}\text{Cruz, Yokota, Barba, Knepley}\]
**Vortex Method**

$t = 300$

- **Incompressible Flow**
  - Gaussian vortex blobs
  - High Re

- **PetFMM**
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- **PetRBF**
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- **Parallelism**
  - MPI
  - GPU

---

*Cruz, Yokota, Barba, Knepley*
Incompressible Flow
  - Gaussian vortex blobs
  - High Re

PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

Parallelism
  - MPI
  - GPU

\^aCruz, Yokota, Barba, Knepley
Vortex Method

$t = 500$

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

\[ ^a \text{Cruz, Yokota, Barba, Knepley} \]

\[ a \]
What can PETSc do?

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

Who uses PETSc?

- Cruz, Yokota, Barba, Knepley

\(^{a}\)Cruz, Yokota, Barba, Knepley
What can PETSc do?

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

Who uses PETSc?

- Cruz, Yokota, Barba, Knepley

\(^{a}\)Cruz, Yokota, Barba, Knepley
What can PETSc do?

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

Who uses PETSc?

\(^a\)Cruz, Yokota, Barba, Knepley
What can PETSc do?

**FEniCS-Apps**

**Rheagen**

- Rheologies
  - Maxwell
  - Grade 2
  - Oldroyd-B
- Stabilization
  - DG
  - SUPG
  - EVSS
  - DEVSS
  - Macroelement
- Automation
  - FIAT (elements)
  - FFC (weak forms)

---

\(^a\)Terrel
What can PETSc do?

- Rheologies
  - Maxwell
  - Grade 2
  - Oldroyd-B
- Stabilization
  - DG
  - SUPG
  - EVSS
  - DEVSS
  - Macroelement
- Automation
  - FIAT (elements)
  - FFC (weak forms)

Who uses PETSc?
What can PETSc do?

Who uses PETSc?

Real-time Surgery

- Brain Surgery
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon

- Laser Thermal Therapy
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem

---

*Warfield, Ferrant, et.al.*
Real-time Surgery

- **Brain Surgery**
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon

- **Laser Thermal Therapy**
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem

---

*aFuentes, Oden, et.al.*
Outline

1 What can PETSc do?

2 What’s New in PETSc?
   - Python Bindings
   - Physics-Based Preconditioning
   - PETSc-GPU
   - FEM

3 Conclusion
Outline

2 What’s New in PETSc?
  - Python Bindings
  - Physics-Based Preconditioning
  - PETSc-GPU
  - FEM
**numpy** is ideal for building Python data structures

- Supports multidimensional arrays
- Easily interfaces with C/C++ and Fortran
- High performance BLAS/LAPACK and functional operations
- Python 2 and 3 compatible
- Used by petsc4py to talk to PETSc
petcs4py provides Python bindings for PETSc

- Provides **ALL** PETSc functionality in a Pythonic way
  - Logging using the Python `with` statement
- Can use Python callback functions
  - `SNESSetFunction()`, `SNESSetJacobian()`
- Manages all memory (creation/destruction)
- Visualization with `matplotlib`
petsc4py Installation

- **Automatic**
  - `pip install --install-options=-user petsc4py`
  - Uses `$PETSC_DIR` and `$PETSC_ARCH`
  - Installed into `$HOME/.local`
  - No additions to PYTHONPATH

- **From Source**
  - `virtualenv python-env`
  - `source ./python-env/bin/activate`
  - Now everything installs into your proxy Python environment
  - `hg clone https://petsc4py.googlecode.com/hg petsc4py-dev`
  - `ARCHFLAGS="-arch x86_64" python setup.py sdist`
  - `ARCHFLAGS="-arch x86_64" pip install dist/petsc4py-1.1.2.tar.gz`
  - **ARCHFLAGS** only necessary on Mac OSX
petsc4py Examples

- externalpackages/petsc4py-1.1/demo/bratu2d/bratu2d.py
  - Solves Bratu equation (SNES ex5) in 2D
  - Visualizes solution with matplotlib

- src/ts/examples/tutorials/ex8.py
  - Solves a 1D ODE for a diffusive process
  - Visualize solution using -vec_view_draw
  - Control timesteps with -ts_max_steps
import sys, petsc4py
petsc4py.init(sys.argv)
from petsc4py import PETSc
import math

# Create the grid
da = PETSc.DA().create([-9], comm=PETSc.COMM_WORLD)
f = da.createGlobalVector()
x = f.duplicate()
J = da.getMatrix(PETSc.Mat.Type.AIJ);

# Create the solver
ts = PETSc.TS().create(PETSc.COMM_WORLD)
ts.setProblemType(PETSc.TS.ProblemType.NONLINEAR)
ts.setType(ts.Type.GL)

# Define the problem
ode = MyODE(da)
ts.setIFunction(node.function, f)
ts.setIJacobian(node.jacobian, J)
class MyODE:
    def function(self, ts, t, x, xdot, f):
        mx = da.getSizes()[0]; hx = 1.0/mx
        (xs, xm) = da.getCorners(); xs = xs[0]; xm = xm[0]
        xx = da.createLocalVector()
        xxdot = da.createLocalVector()
        da.globalToLocal(x, xx)
        da.globalToLocal(xdot, xxdot)
        dt = ts.getTimeStep()
        x0 = ts.getSolution()
        if xs == 0: f[0] = xx[0]/hx; xs = 1;
        if xs+xm >= mx: f[mx-1] = xx[xm-(xs==1)]/hx; xm = xm-(xs==1);
        for i in range(xs, xs+xm-1):
            f[i] = xxdot[i-xs+1]
            + (2.0*xx[i-xs+1] - xx[i-xs] - xx[i-xs+2])/hx
            - hx*math.exp(xx[i-xs+1])
        f.assemble()
ts.setTimeStep(0.1)
ts.setDuration(10, 1.0)
ts.setFromOptions()
x.set(1.0)
ts.solve(x)
What’s New in PETSc?

- Python Bindings
- Physics-Based Preconditioning
- PETSc-GPU
- FEM
MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
  - `VecScatter` and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
  - Generalizes `LocalToGlobal()` mapping
- is composable with **ANY** PETSc solver and preconditioner
  - This can be done recursively
MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
  - *VecScatter* and *MatGetSubMatrix()* for efficiency
- assemble functions/operators over all physics
  - Generalizes *LocalToGlobal()* mapping
- is composable with **ANY** PETSc solver and preconditioner
  - This can be done recursively

FieldSplit provides the **buildings blocks** for multiphysics preconditioning.
Physics-Based Preconditioning

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
  - VecScatter and MatGetSubMatrix() for efficiency

- assemble functions/operators over all physics
  - Generalizes LocalToGlobal() mapping

- is composable with ANY PETSc solver and preconditioner
  - This can be done recursively

Notice that this works in exactly the same manner as
- multiple resolutions (MG, FMM, Wavelets)
- multiple domains (Domain Decomposition)
- multiple dimensions (ADI)
Several varieties of preconditioners can be supported:

- Block Jacobi or Block Gauss-Siedel
- Schur complement
- Block ILU (approximate coupling and Schur complement)
- Dave May’s implementation of Elman-Wathen type PCs which only require actions of individual operator blocks

Notice also that we may have any combination of

- “canned” PCs (ILU, AMG)
- PCs needing special information (MG, FMM)
- custom PCs (physics-based preconditioning, Born approximation)

since we have access to an algebraic interface.
Stokes Example

The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type`
- `fieldsplit_0_pc_type ml`
- `fieldsplit_0_ksp_type preonly`

\[
\begin{pmatrix}
A & B \\
B^T & 0
\end{pmatrix}
\]
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`
The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type multiplicative`
- `fieldsplit_0_pc_type ml`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type jacobi`
- `fieldsplit_1_ksp_type preonly`
The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `fieldsplit_0_pc_type ml`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type none`
- `fieldsplit_1_ksp_type minres`
- `pc_fieldsplit_schur_factorization_type diag`
The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `fieldsplit_0_pc_type ml`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type none`
- `fieldsplit_1_ksp_type minres`
- `pc_fieldsplit_schur_factorization_type lower`
Stokes Example

The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `fieldsplit_0_pc_type ml`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type none`
- `fieldsplit_1_ksp_type minres`
- `pc_field-split_schur_factorization_type upper`

\[
\begin{pmatrix}
\hat{A} & B \\
0 & \hat{S}
\end{pmatrix}
\]
The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `pc_fieldsplit_schur_factorization_type full`

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

What’s New in PETSc?

- Python Bindings
- Physics-Based Preconditioning
- PETSc-GPU
- FEM
Thrust is a CUDA library of parallel algorithms

- Interface similar to C++ Standard Template Library
- Containers (vector) on both host and device
- Algorithms: sort, reduce, scan
- Freely available, part of PETSc configure (-with-thrust-dir)
- Included as part of CUDA 4.0 installation
Cusp is a CUDA library for sparse linear algebra and graph computations

- Builds on data structures in Thrust
- Provides sparse matrices in several formats (CSR, Hybrid)
- Includes some preliminary preconditioners (Jacobi, SA-AMG)
- Freely available, part of PETSc configure (\texttt{--with-cusp-dir})
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
Memory Coherence

PETSc Objects now hold a coherence flag

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PETSC_CUDA_UNALLOCATED</td>
<td>No allocation on the GPU</td>
</tr>
<tr>
<td>PETSC_CUDA_GPU</td>
<td>Values on GPU are current</td>
</tr>
<tr>
<td>PETSC_CUDA_CPU</td>
<td>Values on CPU are current</td>
</tr>
<tr>
<td>PETSC_CUDA_BOTH</td>
<td>Values on both are current</td>
</tr>
</tbody>
</table>

Table: Flags used to indicate the memory state of a PETSc CUDA Vec object.
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 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
PETSc only needs

# Turn on CUDA
--with-cuda

# Specify the CUDA compiler
--with-cudac='nvcc -m64'

# Indicate the location of packages
# --download-* will also work soon
--with-thrust-dir=/PETSc3/multicore/thrust
--with-cusp-dir=/PETSc3/multicore/cusp

# Can also use double precision
--with-precision=single
Example
Driven Cavity Velocity-Vorticity with Multigrid

```
ex50  -da_vec_type seqcusp
   -da_mat_type aijcusp  -mat_no_inode
   -da_grid_x 100  -da_grid_y 100
   -pc_type none  -pc_mg_levels 1
   -preload off  -cuda_synchronize
   -log_summary
```

# Setup types
# Set grid size
# Setup solver
# Setup run
What's New in PETSc?

- Python Bindings
- Physics-Based Preconditioning
- PETSc-GPU
- FEM
Local (analytical)
- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)
- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)
Global and Local

Local (analytical)
- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
  - Largely dim dependent (e.g. quadrature)

Global (topological)
- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)
Local (analytical)
- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)
- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)
Local (analytical)
- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

Global (topological)
- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)
Global and Local

**Local (analytical)**
- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions
- Largely dim dependent (e.g. quadrature)

**Global (topological)**
- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies
- Largely dim independent (e.g. mesh traversal)
Can formulate assembly independent of

- spatial dimension
- element shape
- finite element (discretization)
- weak form (using FEniCS)
Integration

cells = mesh->heightStratum(0);
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector*/
}
/* Aggregate updates */
for (c = cells->begin(); c != cells->end(); ++c) {
    SectionRestrictClosure(coordinates, dm, c, &coords);
    v0, J, invJ, detJ = computeGeometry(coords);
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector */
}
/* Aggregate updates */
Integration

```c
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector*/
}
/* Aggregate updates */
```
Integration

```c
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    SectionRestrictClosure(U, dm, c, &inputVec);
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q] * detJ;
        }
    }
    /* Update output vector */
}
/* Aggregate updates */
```
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector*/
}
/* Aggregate updates */
What's New in PETSc?

Integration

```c
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        realCoords = J*refCoords[q] + v0;
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector */
}
/* Aggregate updates */
```
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector*/
}
/* Aggregate updates */
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            elemVec[f] += basis[q,f]*rhsFunc(realCoords);
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector */
}
/*Aggregate updates*/
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q] * detJ;
        }
    }
    /* Update output vector */
}
/* Aggregate updates */
What's New in PETSc?

Integration

```c
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Transform J */
            for (d = 0; d < dim; ++d)
                for (e = 0; e < dim; ++e)
                    tDerReal[d] += invJ[e,d]*basisDer[q,f,e];
        }
        for (g = 0; g < numBasisFuncs; ++g) {
            for (d = 0; d < dim; ++d)
                for (e = 0; e < dim; ++e)
                    bDerReal[d] += invJ[e,d]*basisDer[q,g,e];
        }
        /* Update element matrix */
        /* Update element vector */
    }
    /* Nonlinear term */
    elemVec[f] *= weight[q]*detJ;
}
/* Update output vector*/
/* Aggregate updates */
```
```c
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Transform J */
            for (g = 0; g < numBasisFuncs; ++g) {
                for (d = 0; d < dim; ++d)
                    elemMat[f][g] += tDerReal[d] * bDerReal[d];
                elemVec[f] += elemMat[f][g] * inputVec[g];
            }
            /* Nonlinear term */
            elemVec[f] *= weight[q] * detJ;
        }
    }
    /* Aggregate updates */
}
```
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector */
}
/* Aggregate updates */
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            elemVec[f] += basis[q,f]*lambda*exp(inputVec[f])
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector */
}
/* Aggregate updates */
for(c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for(q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for(f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector*/
}
/* Aggregate updates */
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q] * detJ;
        }
    }
    SectionRealUpdate(locF, c, elemVec, ADD_VALUES);
}
/* Aggregate updates */
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q] * detJ;
        }
    }
    /* Update output vector*/
}
/* Aggregate updates */
for (c = cells->begin(); c != cells->end(); ++c) {
    /* Compute cell geometry */
    /* Retrieve values from input vector */
    for (q = 0; q < numQuadPoints; ++q) {
        /* Transform coordinates */
        for (f = 0; f < numBasisFuncs; ++f) {
            /* Constant term */
            /* Linear term */
            /* Nonlinear term */
            elemVec[f] *= weight[q]*detJ;
        }
    }
    /* Update output vector */
}
DMLocalToGlobalBegin(dm, locF, INSERT_VALUES, F);
DMLocalToGlobalEnd(dm, locF, INSERT_VALUES, F);
Preliminary system for FEM integration on a GPU

- High order is basically done by others
- Low order much more prevalent in applications
- Use PyCUDA and Mako to generate kernels
- Nearly 100GF on a GTX 285
Outline

1. What can PETSc do?
2. What's New in PETSc?
3. Conclusion
PETSc gives you tools to design and build new Scientific Software from simple pieces.
Why is PETSc cool?

PETSc gives you tools to design and build new Scientific Software from simple pieces.
Your Stuff is more important than Our Stuff
Why is PETSc cool?

Your Stuff is more important than Our Stuff
Why is PETSc cool?

Your Stuff is more important than Our Stuff