PETSc: 1001 Applications

Matthew Knepley

Computation Institute
University of Chicago
Department of Molecular Biology and Physiology
Rush University Medical Center

Department of Applied Mathematics and Computational Science
King Abdullah University of Science and Technology
Apr 5, 2010
What can PETSc do?

- What is PETSc?
- Who uses PETSc?

Value of Design
Outline

1. What can PETSc do?
   - What is PETSc?
   - Who uses PETSc?
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 for the parallel solution of nonlinear algebraic equations

Free
- Download from http://www.mcs.anl.gov/petsc
- Free for everyone, including industrial users

Supported
- 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, Matlab, Julia, and Python
What can PETSc do?

Portable to any parallel system supporting MPI, including:
- Tightly coupled systems
  - Cray XT6, BG/Q, NVIDIA Fermi, K Computer
- 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
The PETSc Team

Bill Gropp
Barry Smith
Satish Balay
Jed Brown
Matt Knepley
Lisandro Dalcin
Hong Zhang
Mark Adams
Toby Issac
Outline

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

Computational Scientists

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

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

Computational Scientists

- CFD
  - Firedrake
  - Fluidity
  - OpenFOAM
  - freeCFD
  - OpenFVM

- MicroMagnetics
  - MagPar

- Fusion
  - XGC
  - BOUT++
  - NIMROD
Who Uses PETSc?

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
  - libMesh
  - MOOSE
  - PETSc-FEM
  - Deal II
  - OOFEM

- Other Solvers
  - Fast Multipole Method (PetFMM)
  - Radial Basis Function Interpolation (PetRBF)
  - Eigensolvers (SLEPc)
  - Optimization (TAO)
What Can We Handle?

- 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 Jugene at Jülich
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL

- PETSc applications have run at 23% of peak (**600 Teraflops**)
  - Jed Brown on NERSC Edison
  - HPGMG code
What can PETSc do?

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 Jugene at Jülich
- PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 23% of peak (600 Teraflops)
- Jed Brown on NERSC Edison
- HPGMG code
What can PETSc do?

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 Jugene at Jülich
- PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 23% of peak (600 Teraflops)
- Jed Brown on NERSC Edison
- HPGMG code
What can PETSc do?

- 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
  - DMPlex mesh management

Who uses PETSc?

- PyLith
- Aagaard, Knepley, Williams

\[ ^a \text{Aagaard, Knepley, Williams} \]
What can PETSc do?

Multiple Mesh Types

- Triangular
- Tetrahedral
- Rectangular
- Hexahedral

Who uses PETSc?
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

*Katz, Speigelman*
Fracture Mechanics

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

- Uses TAO optimization
  - Necessary for linear penalty
  - Backtracking

- No prescribed cracks (movie)
  - Arbitrary crack geometry
  - Arbitrary intersections

- Multiple materials
  - Composite toughness

\({}^a\text{Bourdin}\)
**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

---

\(^a\text{Cruz, Yokota, Barba, Knepley}\)
Vortex Method

$t = 100$

- 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"
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

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

**t = 400**

- **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

*Cruz, Yokota, Barba, Knepley*
Vortex Method

t = 600

- 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

- 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

\[ a \]

\textsuperscript{a}Cruz, Yokota, Barba, Knepley
Vortex Method

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

\(^a\)Cruz, Yokota, Barba, Knepley
Rheologies
- Maxwell
- Grade 2
- Oldroyd-B

Stabilization
- DG
- SUPG
- EVSS
- DEVSS
- Macroelement

Automation
- FIAT (elements)
- FFC (weak forms)

\[ ^{\text{aTerrel}} \]
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?
Real-time Surgery

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

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

---

*a*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

\[\text{Figure 1: 3D volume rendering of in vivo MR-guided LITT delivery in a canine model of prostate.}\]

Contrast-enhanced T1-W MR images have been volume rendered to better visualize the relationship of the target volume and applicator trajectory to the surrounding anatomy. As displayed, the subject was stabilized in the supine position with legs upward. A stainless steel stylet was used to insert the laser catheter consisting of a 700 µm core diameter, 1 cm diameter silica fiber within a 2mm diameter water-cooled catheter (light blue cylinder). A volume rendering of the multi-planar thermal images (in degrees Celsius) is registered and fused with the 3D anatomy to visualize the 3D volume of therapy while an axial slice cut from the principal treatment plane demonstrates a 2D representation of the local heating in that slice. The full field of view shown is 240mm x 240mm (scale on image in mm).

\[a\text{Fuentes, Oden, et.al.}\]
Outline

1. What can PETSc do?

2. Value of Design
   - DA
   - Mesh
   - DMMG
   - PCFieldSplit
Separating the Local from Global, simplifies design, and enables modern solvers.
Separating the Local from Global, simplifies design, and enables modern solvers.
Separating the Local from Global, simplifies design, and enables modern solvers.
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)
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)
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)
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)
Value of Design

- DA
- Mesh
- DMMG
- PCFieldSplit
Ghost Values

To evaluate a local function $f(x)$, each process requires
- its local portion of the vector $x$
- its **ghost values**, bordering portions of $x$ owned by neighboring processes
Residual Evaluation

The **DM** interface is based upon *local* callback functions

- `FormFunctionLocal()`
- `FormJacobianLocal()`

Callbacks are registered using

- `SNESSetDM()`, `TSSetDM()`
- `DMSNESFunctionLocal()`, `DMTSSetJacobianLocal()`

When PETSc needs to evaluate the nonlinear residual \( \mathbf{F}(\mathbf{x}) \),

- Each process evaluates the local residual
- PETSc assembles the global residual automatically
  - Uses `DMLocalToGlobal()` method
Bratu Residual Evaluation

\[ \Delta u + \lambda e^u = 0 \]

```c
ResLocal(DMDALocalInfo *info, PetscScalar **x, PetscScalar **f, void *ctx) {
    for (j = info->ys; j < info->ys+info->ym; ++j) {
        for (i = info->xs; i < info->xs+info->xm; ++i) {
            u = x[j][i];
            if (i==0 || j==0 || i == M || j == N) {
                f[j][i] = 2.0*(hydhx+hxdhy)*u; continue;
            }
            u_xx = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
            u_yy = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
            f[j][i] = u_xx + u_yy - hx*hy*lambda*exp(u);
        }
    }
}
```

`$PETSC_DIR/src/snes/examples/tutorials/ex5.c`
2 Value of Design

- DA
- Mesh
- DMMG
- PCFieldSplit
Mesh Paradigm

The **DMMesh** interface also uses *local* callback functions

- maps between *global* Vec and *local* Vec
- Local vectors are structured using a **PetscSection**

When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Each process evaluates the local residual for each element
- PETSc assembles the global residual automatically
  - `DMLocalToGlobal()` works just as in the structured case
Ghost Values

To evaluate a local function $f(x)$, each process requires

- its local portion of the vector $x$
- its *ghost values*, bordering portions of $x$ owned by neighboring processes
cells = mesh->heightStratum(0);
for(c = cells->begin(); c != cells->end(); ++c) {
    \texttt{<Compute cell geometry>}
    \texttt{<Retrieve values from input vector>}
    for(q = 0; q < numQuadPoints; ++q) {
        \texttt{<Transform coordinates>}
        for(f = 0; f < numBasisFuncs; ++f) {
            \texttt{<Constant term>}
            \texttt{<Linear term>}
            \texttt{<Nonlinear term>}
            elemVec[f] *= weight[q] * detJ;
        }
    }
    \texttt{<Update output vector>}
}
\texttt{<Aggregate updates>}
Integration

cells = mesh->heightStratum(0);
for (c = cells->begin(); c != cells->end(); ++c) {
   coords = mesh->restrict(coordinates, c);
   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>
}
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>
cells = mesh->heightStratum(0);
for (c = cells->begin(); c != cells->end(); ++c) {
    <Compute cell geometry>
    inputVec = mesh->restrict(U, c);
    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>
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>
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) {
        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>
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>
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) {
            elemVec[f] += basis[q,f]*rhsFunc(realCoords);
            <Linear term>
            <Nonlinear term>
            elemVec[f] *= weight[q]*detJ;
        }
    }
    <Update output vector>
}

<Aggregate updates>
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>
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>
      for(d = 0; d < dim; ++d)
        for(e) testDerReal[d] += invJ[e,d]*basisDer[q,e];
      for(g = 0; g < numBasisFuncs; ++g) {
        for(d = 0; d < dim; ++d)
          for(e) basisDerReal[d] += invJ[e,d]*basisDer[g,e];
        elemMat[f,g] += testDerReal[d]*basisDerReal[e];
        elemVec[f] += elemMat[f,g]*inputVec[g];
      }
    }
  }
  <Nonlinear term>
  elemVec[f] *= weight[q]*detJ;
}
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>
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>
            elemVec[f] += basis[q,f]*lambda*exp(inputVec[f])
            elemVec[f] *= weight[q]*detJ;
        }
    }
    <Update output vector>
}
<Aggregate updates>
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>
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;
        }
    }
    mesh->updateAdd(F, c, elemVec);
}
<Aggregate updates>
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>
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>
}
Distribution<Mesh>::completeSection(mesh, F);
Outline

Value of Design

- DA
- Mesh
- DMMG
- PCFieldSplit
Multigrid Paradigm

The **DM** interface uses the *local* callback functions to
- assemble global functions/operators from local pieces
- assemble functions/operators on coarse grids

Then **PCMG** organizes
- control flow for the multilevel solve, and
- projection and smoothing operators at each level.
Multigrid with DM

Allows multigrid with some simple command line options

- \texttt{-pc\_type mg, -pc\_mg\_levels}
- \texttt{-pc\_mg\_type, -pc\_mg\_cycle\_type, -pc\_mg\_galerkin}
- \texttt{-mg\_levels\_1\_ksp\_type, -mg\_levels\_1\_pc\_type}
- \texttt{-mg\_coarse\_ksp\_type, -mg\_coarse\_pc\_type}
- \texttt{-da\_refine, -ksp\_view}

Interface also works with GAMG and 3rd party packages like ML
Value of Design

- DA
- Mesh
- DMMG
- PCFieldSplit
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.
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

Notice that this works in exactly the same manner as

- multiple resolutions (MG, FMM, Wavelets)

- multiple domains (Domain Decomposition)

- multiple dimensions (ADI)
Solver Configuration: No New Code

ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

\[
\begin{pmatrix}
  A & B \\
  B^T & 0
\end{pmatrix}
\]
Solver Configuration: No New Code

**ex62:** $P_2/P_1$ Stokes Problem on Unstructured Mesh

**Block-Jacobi (Exact)**

-ksp_type gmres -pc_type fieldsplit -pc_fieldsplit_type additive
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
-fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi

\[
\begin{pmatrix}
A & 0 \\
0 & I
\end{pmatrix}
\]
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Block-Jacobi (Inexact)

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type additive
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi

\[
\begin{pmatrix}
\hat{A} & 0 \\
0 & I
\end{pmatrix}
\]
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Gauss-Seidel (Inexact)

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type multiplicative
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi

\[
\begin{pmatrix}
\hat{A} & B \\
0 & I
\end{pmatrix}
\]
**Solver Configuration: No New Code**

**ex62**: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Gauss-Seidel (Inexact)

```bash
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 1 -pc_fieldsplit_1_fields 0
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

\[
\begin{pmatrix}
I & B^T \\
0 & \hat{A}
\end{pmatrix}
\]
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Diagonal Schur Complement

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type diag
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_ksp_type minres -fieldsplit_pressure_pc_type none

$$\begin{pmatrix}
\hat{A} & 0 \\
0 & -\hat{S}
\end{pmatrix}$$
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Lower Schur Complement

\[
\begin{pmatrix}
\hat{A} & 0 \\
B^T & \hat{S}
\end{pmatrix}
\]
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Upper Schur Complement

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type upper
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
-fieldsplit_pressure_ksp_type minres -fieldsplit_pressure_pc_type none
```

\[
\begin{pmatrix}
\hat{A} & B \\
\hat{S} & \hat{S}
\end{pmatrix}
\]
Solver Configuration: No New Code

ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Uzawa

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type upper
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
-fieldsplit_pressure_ksp_type richardson
-fieldsplit_pressure_ksp_max_its 1

$$\begin{pmatrix} A & B \\ \hat{S} \end{pmatrix}$$
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Full Schur Complement

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
-fieldsplit_pressure_ksp_rtol 1e-10 -fieldsplit_pressure_pc_type jacobi

\[
\begin{pmatrix}
I & 0 \\
B^T A^{-1} & I
\end{pmatrix}
\begin{pmatrix}
A & 0 \\
0 & S
\end{pmatrix}
\begin{pmatrix}
I & A^{-1} B \\
0 & I
\end{pmatrix}
\]
**Solver Configuration: No New Code**

**ex62:** $P_2/P_1$ Stokes Problem on Unstructured Mesh

**SIMPLE**

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
-fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
-fieldsplit_pressure_ksp_rtol 1e-10 -fieldsplit_pressure_pc_type jacobi
-fieldsplit_pressure_inner_ksp_type preonly
-fieldsplit_pressure_inner_pc_type jacobi
-fieldsplit_pressure_upper_ksp_type preonly
-fieldsplit_pressure_upper_pc_type jacobi
```

\[
\begin{pmatrix}
I & 0 \\
B^T A^{-1} & I
\end{pmatrix}
\begin{pmatrix}
A & 0 \\
0 & B^T D_A^{-1} B
\end{pmatrix}
\begin{pmatrix}
I & D_A^{-1} B \\
0 & I
\end{pmatrix}
\]
ex62: $P_2/P_1$ Stokes Problem on Unstructured Mesh

Least-Squares Commutator

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
-PC_fieldsplit_schur_precondition self
-Fieldsplit_velocity_ksp_type gmres -Fieldsplit_velocity_pc_type lu
-Fieldsplit_pressure_ksp_rtol 1e-5 -Fieldsplit_pressure_pc_type lsc

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & \hat{S}_{LSC} \end{pmatrix} \begin{pmatrix} I & A^{-1} B \\ 0 & I \end{pmatrix}$$
ex31: $P_2/P_1$ Stokes Problem with Temperature on Unstructured Mesh

Additive Schwarz + Full Schur Complement

-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type additive
-pc_fieldsplit_0_fields 0,1 -pc_fieldsplit_1_fields 2
-fieldsplit_0_ksp_type fgmres -fieldsplit_0_pc_type fieldsplit
-fieldsplit_0_pc_fieldsplit_type schur
-fieldsplit_0_pc_fieldsplit_schur_factorization_type full
-fieldsplit_0_fieldsplit_velocity_ksp_type preonly
-fieldsplit_0_fieldsplit_velocity_pc_type lu
-fieldsplit_0_fieldsplit_pressure_ksp_rtol 1e-10
-fieldsplit_0_fieldsplit_pressure_pc_type jacobi
-fieldsplit_temperature_ksp_type preonly
-fieldsplit_temperature_pc_type lu

\[
\begin{pmatrix}
I & 0 \\
B^T A^{-1} & I
\end{pmatrix}
\begin{pmatrix}
\hat{A} & 0 \\
0 & \hat{S}
\end{pmatrix}
\begin{pmatrix}
I & A^{-1} B \\
0 & I
\end{pmatrix}
\begin{pmatrix}
0 \\
L_T
\end{pmatrix}
\]
**Solver Configuration: No New Code**

**ex31**: $P_2/P_1$ Stokes Problem with Temperature on Unstructured Mesh

**Upper Schur Comp. + Full Schur Comp. + Least-Squares Comm.**

```plaintext
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_0_fields 0,1 -pc_fieldsplit_1_fields 2
-pc_fieldsplit_schur_factorization_type upper
-fieldsplit_0_ksp_type fgmres -fieldsplit_0_pc_type fieldsplit
-fieldsplit_0_pc_fieldsplit_type schur
-fieldsplit_0_pc_fieldsplit_schur_factorization_type full
-fieldsplit_0_fieldsplit_velocity_ksp_type preonly
-fieldsplit_0_fieldsplit_velocity_pc_type lu
-fieldsplit_0_fieldsplit_pressure_ksp_rtol 1e-10
-fieldsplit_0_fieldsplit_pressure_pc_type jacobi
-fieldsplit_temperature_ksp_type gmres
-fieldsplit_temperature_pc_type lsc

$$\begin{pmatrix}
I & 0 & I & A^{-1}B \\
B^TA^{-1} & I & 0 & \tilde{S} \\
0 & 0 & I & 0 \\
0 & 0 & 0 & \tilde{S}_{LSC}
\end{pmatrix}$$
```

M. Knepley (UC)  PETSc  KAUST  39 / 40
Conclusion

With good code design, PETSc can make Linear Algebra
With good code design, PETSc can make Solving Equations
Conclusion

With good code design, PETSc can make Mesh and Data Management
Conclusion

With good code design, PETSc can make Multigrid
With good code design, PETSc can make Multiphysics
Conclusion

With good code design, PETSc can make FMM
Conclusion

With good code design, PETSc can make Scientific Computing easy.