

The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Computation Institute
University of Chicago

Workshop for Advancing Numerical Modeling
of Mantle Convection and Lithospheric Dynamics
UC Davis, CA July 9-11, 2008



Outline

- 1 What the Heck is PETSc?
 - What is PETSc?
 - Who uses and develops PETSc?
 - How can I get PETSc?
 - How do I get more help?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

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)

The Role of PETSc

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

What is PETSc?

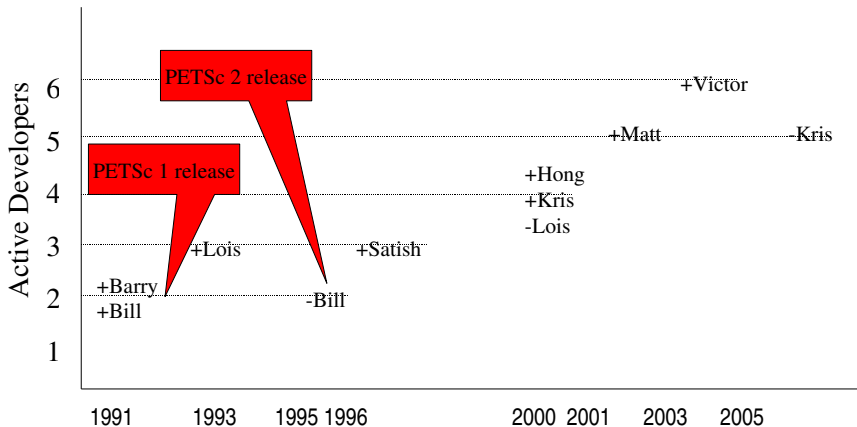
A freely available and supported research code

- Download from <http://www.mcs.anl.gov/petsc>
- 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 is PETSc?

- Portable to any parallel system supporting MPI, including:
 - Tightly coupled systems
 - Cray T3E, SGI Origin, IBM SP, HP 9000, Sub Enterprise
 - Loosely coupled systems, such as networks of workstations
 - Compaq, HP, IBM, SGI, Sun, PCs running Linux or Windows
- PETSc History
 - Begun September 1991
 - Over 20,000 downloads since 1995 (version 2), currently 300 per month
- PETSc Funding and Support
 - Department of Energy
 - SciDAC, MICS Program, INL Reactor Program
 - National Science Foundation
 - CIG, CISE, Multidisciplinary Challenge Program

Timeline



What Can We Handle?

- PETSc has run implicit problems with over **500 million** unknowns
 - <http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf>

What Can We Handle?

- PETSc has run implicit problems with over **500 million** unknowns
 - <http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf>
- PETSc has run on over **27,580** processors efficiently
 - PFLOTRAN on the Cray XT4 Jaguar at ORNL

What Can We Handle?

- PETSc has run implicit problems with over **500 million** unknowns
 - <http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf>
- PETSc has run on over **27,580** processors efficiently
 - PFLOTRAN on the Cray XT4 Jaguar at ORNL
- PETSc applications have run at **2 Teraflops**
 - LANL PFLOTRAN code

Who Uses PETSc?

- **Computational Scientists**
 - PyLith (TECTON), Underworld, Columbia group, PFLOTRAN
- **Algorithm Developers**
 - Iterative methods and Preconditioning researchers
- **Package Developers**
 - SLEPc, TAO, PETSc-FEM, MagPar, StGermain, DealII

The PETSc Team



Bill Gropp



Barry Smith



Satish Balay



Dinesh Kaushik



Kris Buschelman



Matt Knepley



Hong Zhang



Victor Eijkhout



Lois McInnes

Downloading PETSc

- The latest tarball is on the PETSc site
 - `ftp://ftp.mcs.anl.gov/pub/petsc/petsc.tar.gz`
 - We no longer distribute patches (everything is in the distribution)
- There is a Debian package
- There is a FreeBSD Port
- There is a Mercurial development repository

Cloning PETSc

- The full development repository is open to the public
 - <http://petsc.cs.iit.edu/petsc/petsc-dev>
 - <http://petsc.cs.iit.edu/petsc/BuildSystem>
- Why is this better?
 - You can clone to any release (or any specific ChangeSet)
 - You can easily rollback changes (or releases)
 - You can get fixes from us the same day
- We also make release repositories available
 - <http://petsc.cs.iit.edu/petsc/petsc-release-2.3.3>

Automatic Downloads

- Starting in 2.2.1, some packages are automatically
 - Downloaded
 - Configured and Built (in `$PETSC_DIR/externalpackages`)
 - Installed with PETSc
- Currently works for
 - PETSc documentation utilities (Sowing, lgrind, c2html)
 - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
 - MPICH, MPE, LAM
 - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
 - MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
 - BLOPEX, FFTW, SPRNG
 - Prometheus, HYPRE, ML, SPAI
 - Sundials
 - Triangle, TetGen
 - FIAT, FFC, Generator
 - Boost

Getting More Help

- <http://www.mcs.anl.gov/petsc>
- Hyperlinked documentation
 - Manual
 - Manual pages for every method
 - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov
- High profile users
 - Marc Spiegelman
 - Richard Katz
 - Brad Aagaard
 - David Keyes

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

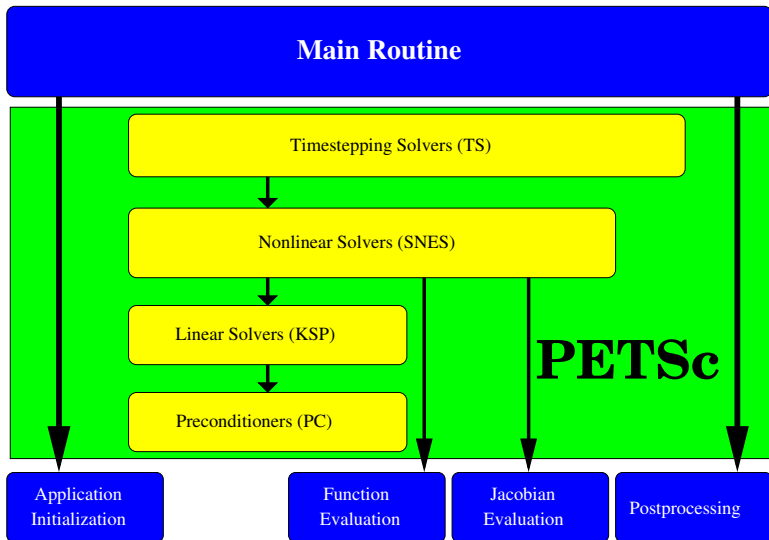
5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

Flow Control for a PETSc Application



SNES Paradigm

The SNES interface is based upon callback functions

- `FormFunction()`, **set by** `SNESSetFunction()`
- `FormJacobian()`, **set by** `SNESSetJacobian()`

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

- Solver calls the **user's** function
- User function gets application state through the `ctx` variable
 - PETSc *never* sees application data

Topology Abstractions

- DA
 - Abstracts Cartesian grids in any dimension
 - Supports stencils, communication, reordering
 - Nice for simple finite differences
- Mesh
 - Abstracts general topology in any dimension
 - Also supports partitioning, distribution, and global orders
 - Allows arbitrary element shapes and discretizations

Assembly Abstractions

- DM
 - Abstracts the logic of multilevel (multiphysics) methods
 - Manages allocation and assembly of local and global structures
 - Interfaces to `DMMG` solver

- `Section`
 - Abstracts functions over a topology
 - Manages allocation and assembly of local and global structures
 - Will merge with `DM` somehow

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

DA Paradigm

The DA interface is based upon *local* callback functions

- `FormFunctionLocal()`, **set by** `DASetLocalFunction()`
- `FormJacobianLocal()`, **set by** `DASetLocalJacobian()`

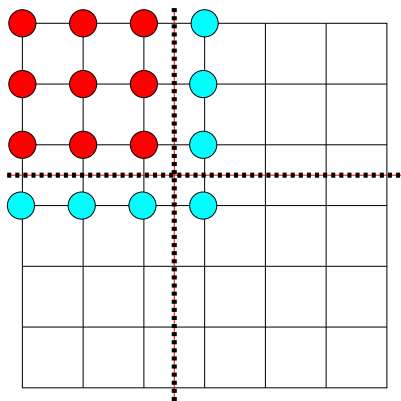
When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Each process evaluates the local residual
- PETSc assembles the global residual automatically

Ghost Values

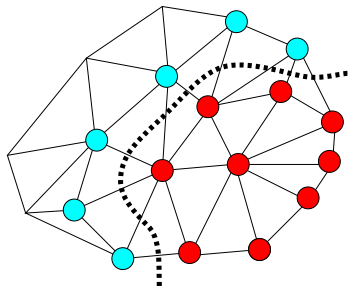
To evaluate a local function $f(x)$, each process requires

- its local portion of the vector x
- its **ghost values**, bordering portions of x owned by neighboring processes



● Local Node

● Ghost Node



DA Global Numberings

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

DA Global vs. Local Numbering

- **Global:** Each vertex belongs to a unique process and has a unique id
- **Local:** Numbering includes **ghost** vertices from neighboring processes

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

DA Local Function

The user provided function which calculates the nonlinear residual in 2D has signature

```
PetscErrorCode (*lfunc) (DALocalInfo *info,  
    PetscScalar **x, PetscScalar **r, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution

- Notice that it is a multidimensional array

`r`: The residual

`ctx`: The user context passed to `DASetLocalFunction()`

The local DA function is activated by calling

```
SNESSetFunction(snes, r, SNESDAFormFunction, ctx)
```

Bratu Residual Evaluation

$$\Delta u + \lambda e^u = 0$$

```

BratuResidualLocal(DALocalInfo *info,Field **x,Field **f)
{
  /* Not Shown: Handle boundaries */
  /* Compute over the interior points */
  for(j = info->ys; j < info->xs+info->ym; j++) {
    for(i = info->xs; i < info->ys+info->xm; i++) {
      u          = x[j][i];
      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);
    }
  }
}

```

\$PETCS_DIR/src/snes/examples/tutorials/ex5.c

DA Local Jacobian

The user provided function which calculates the Jacobian in 2D has signature

```
PetscErrorCode (*lfunc) (DALocalInfo *info,  
                          PetscScalar **x, Mat J, void *ctx)
```

info: All layout and numbering information

x: The current solution

J: The Jacobian

ctx: The user context passed to `DASetLocalFunction()`

The local DA function is activated by calling

```
SNESSetJacobian(snes, J, J, SNESDAComputeJacobian,  
               ctx)
```

A DA is more than a Mesh

A DA contains **topology**, **geometry**, and an implicit Q1 **discretization**.

It is used as a template to create

- Vectors (functions)
- Matrices (linear operators)

DA Vectors

- The DA object contains only layout (topology) information
 - All field data is contained in PETSc `Vecs`
- Global vectors are parallel
 - Each process stores a unique local portion
 - `DACreateGlobalVector(DA da, Vec *gvec)`
- Local vectors are sequential (and usually temporary)
 - Each process stores its local portion plus ghost values
 - `DACreateLocalVector(DA da, Vec *lvec)`
 - includes ghost values!

Updating Ghosts

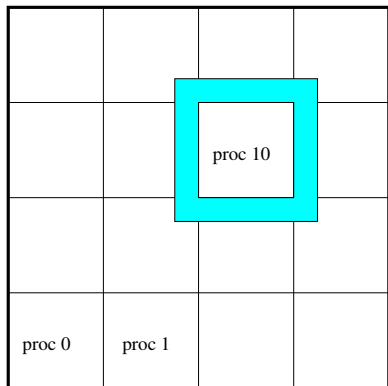
Two-step process enables overlapping computation and communication

- `DAGlobalToLocalBegin(da, gvec, mode, lvec)`
 - `gvec` provides the data
 - `mode` is either `INSERT_VALUES` or `ADD_VALUES`
 - `lvec` holds the local and ghost values
- `DAGlobalToLocalEnd(da, gvec, mode, lvec)`
 - Finishes the communication

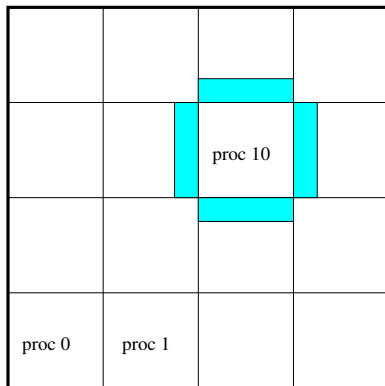
The process can be reversed with `DALocalToGlobal()`.

DA Stencils

Both the **box** stencil and **star** stencil are available.



Box Stencil



Star Stencil

Setting Values on Regular Grids

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n,  
    MatStencil idxn[], values[], mode)
```

- Each row or column is actually a `MatStencil`
 - This specifies grid coordinates and a component if necessary
 - Can imagine for unstructured grids, they are *vertices*
- The values are the same logically dense block in rows and columns

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

Mesh Paradigm

The Mesh interface also uses *local* callback functions

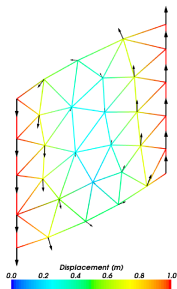
- maps between global `Vec` and local `Vec (Section)`
- provides `Complete()` which generalizes `LocalToGlobal()`

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

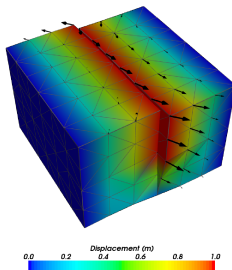
- Each process evaluates the local residual for each element
- PETSc assembles the global residual automatically

Multiple Mesh Types

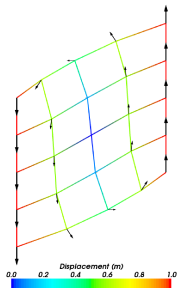
Triangular



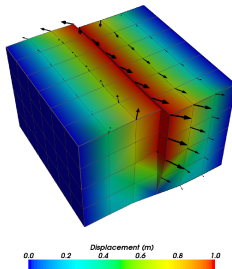
Tetrahedral



Rectangular

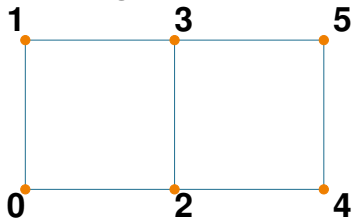


Hexahedral

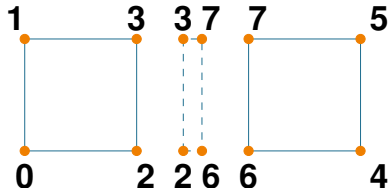
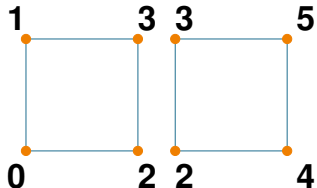
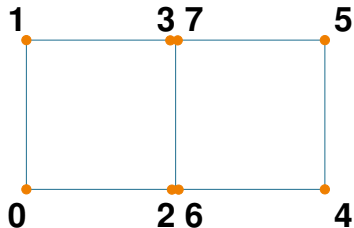


Cohesive Cells

Original Mesh



Mesh with Cohesive Cell



Exploded view of meshes

Cohesive Cells

Cohesive cells are used to enforce slip conditions on a fault

- Demand complex mesh manipulation
 - We allow specification of only fault vertices
 - Must “sew” together on output
- Use Lagrange multipliers to enforce constraints
 - Forces illuminate physics
- Allow different fault constitutive models
 - Simplest is enforced slip
 - Can write a general relation

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

DMMG Paradigm

The DMMG interface uses the *local* DA/Mesh callback functions to

- assemble global functions/operators from local pieces
- assemble functions/operators on coarse grids

DMMG relies upon DM to organize the

- assembly
- coarsening/refinement

while it organizes the control flow for the multilevel solve.

DMMG Integration with SNES

- DMMG supplies global residual and Jacobian to SNES
 - User supplies local version to DMMG
 - The `Rhs_*()` and `Jac_*()` functions in the example
- Allows automatic parallelism
- Allows grid hierarchy
 - Enables multigrid once interpolation/restriction is defined
- Paradigm is developed in unstructured work
 - Notice we have to scatter into contiguous global vectors (initial guess)
- Handle Neumann BC using `DMMGSetNullSpace()`

Structured Meshes

The DMMG allows multigrid which some simple options

- `-dmmg_nlevels`, `-dmmg_view`
- `-pc_mg_type`, `-pc_mg_cycle_type`
- `-mg_levels_1_ksp_type`, `-dmmg_levels_1_pc_type`
- `-mg_coarse_ksp_type`, `-mg_coarse_pc_type`

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

MultiPhysics Paradigm

The PCFieldSplit interface uses the `VecScatter` objects to

- extract functions/operators corresponding to each physics
- assemble functions/operators over all physics

Notice that this works in exactly the same manner for

- multiple resolutions (MG, Wavelets)
- multiple domains (Domain Decomposition)
- multiple dimensions (ADI)

Preconditioning

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

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

<http://www.fenics.org/fiat>

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

User can build arbitrary elements specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project, as is the PETSc Sieve module

FFC is a compiler for variational forms.

Here is a mixed-form Poisson equation:

$$a((\tau, \mathbf{w}), (\sigma, \mathbf{u})) = L((\tau, \mathbf{w})) \quad \forall (\tau, \mathbf{w}) \in V$$

where

$$a((\tau, \mathbf{w}), (\sigma, \mathbf{u})) = \int_{\Omega} \tau \sigma - \nabla \cdot \tau \mathbf{u} + \mathbf{w} \nabla \cdot \mathbf{u} \, dx$$

$$L((\tau, \mathbf{w})) = \int_{\Omega} \mathbf{w} f \, dx$$

FFC is a compiler for variational forms.

```
shape = "triangle"
BDM1 = FiniteElement("Brezzi-Douglas-Marini", shape, 1)
DG0 = FiniteElement("Discontinuous Lagrange", shape, 0)

element = BDM1 + DG0
(tau, w) = TestFunctions(element)
(sigma, u) = TrialFunctions(element)

f = Function(DG0)

a = (dot(tau, sigma) - div(tau)*u + w*div(sigma))*dx
L = w*f*dx
```

FFC is a compiler for variational forms.

Here is a discontinuous Galerkin formulation of the Poisson equation:

$$a(v, u) = L(v) \quad \forall v \in V$$

where

$$\begin{aligned} a(v, u) &= \int_{\Omega} \nabla u \cdot \nabla v \, dx \\ &+ \sum_S \int_S - \langle \nabla v \rangle \cdot [[u]]_n - [[v]]_n \cdot \langle \nabla u \rangle - (\alpha/h)vu \, dS \\ &+ \int_{\partial\Omega} -\nabla v \cdot [[u]]_n - [[v]]_n \cdot \nabla u - (\gamma/h)vu \, ds \\ L(v) &= \int_{\Omega} vf \, dx \end{aligned}$$

FFC is a compiler for variational forms.

```

DG1 = FiniteElement("Discontinuous Lagrange", shape, 1)
v = TestFunctions(DG1)
u = TrialFunctions(DG1)
f = Function(DG1)
g = Function(DG1)
n = FacetNormal("triangle")
h = MeshSize("triangle")
a = dot(grad(v), grad(u))*dx
  - dot(avg(grad(v)), jump(u, n))*dS
  - dot(jump(v, n), avg(grad(u)))*dS
  + alpha/h*dot(jump(v, n) + jump(u, n))*dS
  - dot(grad(v), jump(u, n))*ds
  - dot(jump(v, n), grad(u))*ds
  + gamma/h*v*u*ds
L = v*f*dx + v*g*ds

```

Outline

1 What the Heck is PETSc?

2 SNES

3 DA

4 Mesh

5 DMMG

6 PCFieldSplit

7 FEniCS Tools

8 Conclusions

Conclusions

PETSc can help you

- easily construct a code to test your ideas
- scale an existing code to large or distributed machines
- incorporate more scalable or higher performance algorithms
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
 - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
- incorporate more scalable or higher performance algorithms
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
 - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
 - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
 - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
 - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
 - Such as domain decomposition or multigrid
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
 - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
 - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
 - Such as domain decomposition or multigrid
- tune your code to new architectures
 - Using profiling tools and specialized implementations