PETSc Tutorial

Numerical Software Libraries for the Scalable Solution of PDEs

Satish Balay
Kris Buschelman
Bill Gropp
Lois Curfman McInnes
Barry Smith

Mathematics and Computer Science Division
Argonne National Laboratory
http://www.mcs.anl.gov/petsc

Intended for use with version 2.0.29 of PETSc
Tutorial Objectives

• Introduce the Portable, Extensible Toolkit for Scientific Computation (PETSc)
• Demonstrate how to write a complete parallel implicit PDE solver using PETSc
• Learn about PETSc interfaces to other packages
• How to learn more about PETSc
The Role of PETSc

• Developing parallel, non-trivial 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.
What is PETSc?

- A freely available and supported research code
  - Available via http://www.mcs.anl.gov/petsc
  - Hyperlinked documentation and manual pages for all routines
  - Many tutorial-style examples
  - Support via email: petsc-maint@mcs.anl.gov
  - Usable from Fortran 77/90, C, and C++

- Portable to any parallel system supporting MPI, including
  - Tightly coupled systems
    - Cray T3E, SGI Origin, IBM SP, HP 9000, Sun Enterprise
  - Loosely coupled systems, e.g., networks of workstations
    - Compaq, HP, IBM, SGI, Sun
    - PCs running Linux or NT

- PETSc history
  - Begun in September 1991
  - Now: over 4,000 downloads of version 2.0

- PETSc funding and support
  - Department of Energy, MICS Program – DOE2000
  - National Science Foundation, Multidisciplinary Challenge Program, CISE
PETSc Concepts

• How to specify the mathematics of the problem
  – Data objects
    • vectors, matrices

• How to solve the problem
  – Solvers
    • linear, nonlinear, and time stepping (ODE) solvers

• Parallel computing complications
  – Parallel data layout
    • structured and unstructured meshes
Tutorial Topics

- **Getting started**
  - sample results
  - programming paradigm

- **Data objects**
  - vectors (e.g., field variables)
  - matrices (e.g., sparse Jacobians)

- **Viewers**
  - object information
  - visualization

- **Solvers**
  - linear
  - nonlinear
  - timestepping (and ODEs)

- **Data layout and ghost values**
  - structured and unstructured mesh problems
  - partitioning and coloring

- **Putting it all together**
  - a complete example

- **Debugging and error handling**

- **Profiling and performance tuning**

- **Extensibility issues**

- **Using PETSc with other software packages**
Tutorial Topics:
Using PETSc with Other Packages

- **PVODE** – ODE integrator

- **ILUDTP** – drop tolerance ILU

- **ParMETIS** – parallel partitioner

- **Overture** – composite mesh PDE package

- **SAMRAI** – AMR package

- **SPAI** – sparse approximate inverse preconditioner

- **Matlab**
  - http://www.mathworks.com

- **TAO** – optimization software
Tutorial Approach

From the perspective of an application programmer:

- **Beginner**
  - basic functionality, intended for use by most programmers

  Only in this tutorial

  beginner

- **Intermediate**
  - selecting options, performance evaluation and tuning

  2

  intermediate

- **Advanced**
  - user-defined customization of algorithms and data structures

  3

  advanced

- **Developer**
  - advanced customizations, intended primarily for use by library developers

  4

  developer
Incremental Application Improvement

- **Beginner**
  - Get the application “up and walking”

- **Intermediate**
  - Experiment with options
  - Determine opportunities for improvement

- **Advanced**
  - Extend algorithms and/or data structures *as needed*

- **Developer**
  - Consider interface and efficiency issues for integration and interoperability of multiple toolkits

- **Full tutorials available at** [http://www.mcs.anl.gov/petsc/docs/tutorials](http://www.mcs.anl.gov/petsc/docs/tutorials)
Structure of PETSc

PETSc PDE Application Codes

- ODE Integrators
- Visualization
- Nonlinear Solvers, Unconstrained Minimization
- Interface
- Linear Solvers
- Preconditioners + Krylov Methods
- Grid Management
- Object-Oriented Matrices, Vectors, Indices
- Profiling Interface

Computation and Communication Kernels
MPI, MPI-IO, BLAS, LAPACK
# PETSc Numerical Components

<table>
<thead>
<tr>
<th>Nonlinear Solvers</th>
<th>Time Steppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-based Methods</td>
<td>Other</td>
</tr>
<tr>
<td>Line Search</td>
<td>Trust Region</td>
</tr>
<tr>
<td>Euler</td>
<td>Backward Euler</td>
</tr>
<tr>
<td>Pseudo Time Stepping</td>
<td>Other</td>
</tr>
</tbody>
</table>

## Krylov Subspace Methods

<table>
<thead>
<tr>
<th>Preconditioners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive Schwartz</td>
</tr>
<tr>
<td>LU (Sequential only)</td>
</tr>
</tbody>
</table>

## Matrices

<table>
<thead>
<tr>
<th>Distributed Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Sparse Row (AIJ)</td>
</tr>
</tbody>
</table>

## Index Sets

<table>
<thead>
<tr>
<th>Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices</td>
</tr>
</tbody>
</table>
Flow of Control for PDE Solution

Main Routine

Timestepping Solvers (TS)

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

PETSc

Application Initialization

Function Evaluation

Jacobian Evaluation

Post-Processing

PC

KSP

User code

PETSc code
Flow of Control for PDE Solution

Main Routine

- Overture
- SAMRAI

PETSc

- Timestepping Solvers (TS)
- Nonlinear Solvers (SNES)
- Linear Solvers (SLES)
- PC
- KSP

Application Initialization
- SPAI
- ILUDTP

Function Evaluation

Jacobian Evaluation

Post-Processing
- PVODE

Other Tools

User code
- PETSc code
- Other Tools
Levels of Abstraction in Mathematical Software

• Application-specific interface
  – Programmer manipulates objects associated with the application

• High-level mathematics interface
  – Programmer manipulates mathematical objects, such as PDEs and boundary conditions

• Algorithmic and discrete mathematics interface
  – Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)

• Low-level computational kernels
  – e.g., BLAS-type operations
Basic PETSc Components

• Data Objects
  – Vec (vectors) and Mat (matrices)
  – Viewers

• Solvers
  – Linear Systems
  – Nonlinear Systems
  – Timestepping

• Data Layout and Ghost Values
  – Structured Mesh
  – Unstructured Mesh
PETSc Programming Aids

- **Correctness Debugging**
  - Automatic generation of tracebacks
  - Detecting memory corruption and leaks
  - Optional user-defined error handlers

- **Performance Debugging**
  - Integrated profiling using `-log_summary`
  - Profiling by stages of an application
  - User-defined events
The PETSc Programming Model

• **Goals**
  – Portable, runs everywhere
  – Performance
  – Scalable parallelism

• **Approach**
  – Distributed memory, “shared-nothing”
    • Requires only a compiler (single node or processor)
    • Access to data on remote machines through MPI
  – Can still exploit “compiler discovered” parallelism on each node (e.g., SMP)
  – Hide within parallel objects the details of the communication
  – User orchestrates communication at a higher abstract level than message passing
Collectivity

• MPI communicators (MPI_Comm) specify collectivity (processors involved in a computation)
• All PETSc creation routines for solver and data objects are collective with respect to a communicator, e.g.,
  VecCreate(MPI_Comm comm, int m, int M, Vec *x)
• Some operations are collective, while others are not, e.g.,
  – collective: VecNorm()
  – not collective: VecGetLocalSize()
• If a sequence of collective routines is used, they **must** be called in the same order on each processor
Hello World

```
#include "petsc.h"
int main( int argc, char *argv[] )
{
    PetscInitialize( &argc, &argv, NULL, NULL );
    PetscPrintf( PETSC_COMM_WORLD, "Hello World\n");
    PetscFinalize();
    return 0;
}
```
Hello World (Fortran)

```fortran
program main
integer ierr, rank
#include "include/finclude/petsc.h"
call PetscInitialize( PETSC_NULL_CHARACTER, ierr )
call MPI_Comm_rank( PETSC_COMM_WORLD, rank, ierr )
if (rank .eq. 0) then
   print *, 'Hello World'
endif
call PetscFinalize(ierr)
end
```
```
#include "petsc.h"
int main( int argc, char *argv[] )
{
    int rank;
    PetscInitialize( &argc, &argv,
                     NULL, NULL );

    MPI_Comm_rank( PETSC_COMM_WORLD, &rank );
    PetscSynchronizedPrintf( PETSC_COMM_WORLD,
                             "Hello World from %d\n", rank);
    PetscFinalize();
    return 0;
}
```
Solver Definitions: For Our Purposes

- **Explicit**: Field variables are updated using neighbor information (no global linear or nonlinear solves)
- **Semi-implicit**: Some subsets of variables (e.g., pressure) are updated with global solves
- **Implicit**: Most or all variables are updated in a single global linear or nonlinear solve
Focus On Implicit Methods

- Explicit and semi-explicit are easier cases
- No direct PETSc support for
  - ADI-type schemes
  - spectral methods
  - particle-type methods
Numerical Methods Paradigm

- Encapsulate the latest numerical algorithms in a consistent, application-friendly manner
- Use mathematical and algorithmic objects, not low-level programming language objects
- Application code focuses on mathematics of the global problem, not parallel programming details
Data Objects

- **Vectors (Vec)**
  - focus: field data arising in nonlinear PDEs
- **Matrices (Mat)**
  - focus: linear operators arising in nonlinear PDEs (i.e., Jacobians)

- Object creation
- Object assembly
- Setting options
- Viewing
- User-defined customizations
Vectors

- Fundamental objects for storing field solutions, right-hand sides, etc.
- `VecCreateMPI(..., Vec *)`
  - `MPI_Comm` - processors that share the vector
  - number of elements local to this processor
  - total number of elements
- Each process locally owns a subvector of contiguously numbered global indices
Vector Assembly

• VecSetValues(Vec,...)
  – number of entries to insert/add
  – indices of entries
  – values to add
  – mode: [INSERT_VALUES,ADD_VALUES]

• VecAssemblyBegin(Vec)

• VecAssemblyEnd(Vec)
Parallel Matrix and Vector Assembly

- Processors may generate any entries in vectors and matrices
- Entries need not be generated on the processor on which they ultimately will be stored
- PETSc automatically moves data during the assembly process if necessary
# Selected Vector Operations

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecAXPY(Scalar *a, Vec x, Vec y)</td>
<td>y = y + a*x</td>
</tr>
<tr>
<td>VecAYPX(Scalar *a, Vec x, Vec y)</td>
<td>y = x + a*y</td>
</tr>
<tr>
<td>VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)</td>
<td>w = a*x + y</td>
</tr>
<tr>
<td>VecScale(Scalar *a, Vec x)</td>
<td>x = a*x</td>
</tr>
<tr>
<td>VecCopy(Vec x, Vec y)</td>
<td>y = x</td>
</tr>
<tr>
<td>VecPointwiseMult(Vec x, Vec y, Vec w)</td>
<td>w_i = x_i *y_i</td>
</tr>
<tr>
<td>VecMax(Vec x, int *idx, double *r)</td>
<td>r = max x_i</td>
</tr>
<tr>
<td>VecShift(Scalar *s, Vec x)</td>
<td>x_i = s+x_i</td>
</tr>
<tr>
<td>VecAbs(Vec x)</td>
<td>x_i =</td>
</tr>
<tr>
<td>VecNorm(Vec x, NormType type, double *r)</td>
<td>r =</td>
</tr>
</tbody>
</table>
Simple Example Programs

Location:  petsc/src/sys/examples/tutorials/

- ex2.c - synchronized printing

Location:  petsc/src/vec/examples/tutorials/

- ex1.c, ex1f.F, ex1f90.F - basic vector routines
- ex3.c, ex3f.F - parallel vector layout

And many more examples ...

- on-line exercise

data objects: vectors

beginner
Sparse Matrices

- Fundamental objects for storing linear operators (e.g., Jacobians)
- `MatCreateMPIAIJ(…, Mat *)`
  - `MPI_Comm` - processors that share the matrix
  - number of local rows and columns
  - number of global rows and columns
  - optional storage pre-allocation information
Parallel Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global rows.

```
proc 0
proc 1
proc 2
proc 3
proc 4
```

\{ proc 3: locally owned rows \}

MatGetOwnershipRange(Mat A, int *rstart, int *rend)

- `rstart`: first locally owned row of global matrix
- `rend -1`: last locally owned row of global matrix
Matrix Assembly

- MatSetValues(Mat, ...)
  - number of rows to insert/add
  - indices of rows and columns
  - number of columns to insert/add
  - values to add
  - mode: [INSERT_VALUES, ADD_VALUES]

- MatAssemblyBegin(Mat)
- MatAssemblyEnd(Mat)
Blocked Sparse Matrices

- For multi-component PDEs
- MatCreateMPIBAIJ(…,Mat *)
  - MPI_Comm - processors that share the matrix
  - block size
  - number of local rows and columns
  - number of global rows and columns
  - optional storage pre-allocation information
Blocking: Performance Benefits

More issues and details discussed in *Performance Tuning* section

- 3D compressible Euler code
- Block size 5
- IBM Power2
Viewers

- Printing information about solver and data objects
- Visualization of field and matrix data
- Binary output of vector and matrix data
Viewer Concepts

• Information about PETSc objects
  – runtime choices for solvers, nonzero info for matrices, etc.
• Data for later use in restarts or external tools
  – vector fields, matrix contents
  – various formats (ASCII, binary)
• Visualization
  – *simple* x-window graphics
    • vector fields
    • matrix sparsity structure
Viewing Vector Fields

- **VecView(Vec x, Viewer v);**

- **Default viewers**
  - ASCII (sequential): VIEWER_STDOUT_SELF
  - ASCII (parallel): VIEWER_STDOUT_WORLD
  - X-windows: VIEWER_DRAW_WORLD

- **Default ASCII formats**
  - VIEWER_FORMAT_ASCII_DEFAULT
  - VIEWER_FORMAT_ASCII_MATLAB
  - VIEWER_FORMAT_ASCII_COMMON
  - VIEWER_FORMAT_ASCII_INFO
  - etc.

Solution components, using runtime option
-snes_vecmonitor

- velocity: $u$
- velocity: $v$
- vorticity: $\zeta$
- temperature: $T$
Viewing Matrix Data

- `MatView(Mat A, Viewer v);`
- Runtime options available after matrix assembly
  - `-mat_view_info`
    - info about matrix assembly
  - `-mat_view_draw`
    - sparsity structure
  - `-mat_view`
    - data in ASCII
  - etc.
Solvers: Usage Concepts

Solver Classes

• Linear (SLES)
• Nonlinear (SNES)
• Timestepping (TS)

Usage Concepts

• Context variables
• Solver options
• Callback routines
• Customization
Linear PDE Solution

PETSc

Main Routine

PETSc code

Linear Solvers (SLES)

PC

KSP

Application Initialization

Evaluation of $A$ and $b$

Post-Processing

User code

PETSc code

Beginner

Solvers: linear
Linear Solvers

**Goal:** Support the solution of linear systems, \( Ax=b, \)

particularly for sparse, parallel problems arising within PDE-based models

User provides:
- Code to evaluate \( A, b \)
Sample Linear Application:
Exterior Helmholtz Problem

\[- \nabla^2 u - k^2 u = 0\]

\[
\lim_{r \to \infty} r^{1/2} \left( \frac{\partial u}{\partial r} + iku \right) = 0
\]

Collaborators: H. M. Atassi, D. E. Keyes,
L. C. McInnes, R. Susan-Resiga

solvers: linear
Helmholtz: The Linear System

- Logically regular grid, parallelized with DAs
- Finite element discretization (bilinear quads)
- Nonreflecting exterior BC (via DtN map)
- Matrix sparsity structure (option: `-mat_view_draw`)

![Natural ordering]
![Close-up]
![Nested dissection ordering]

solvers: linear

beginner
Linear Solvers (SLES)

**SLES: Scalable Linear Equations Solvers**

- Application code interface
- Choosing the solver
- Setting algorithmic options
- Viewing the solver
- Determining and monitoring convergence
- Providing a different preconditioner matrix
- Matrix-free solvers
- User-defined customizations
Context Variables

- Are the key to solver organization
- Contain the complete state of an algorithm, including
  - parameters (e.g., convergence tolerance)
  - functions that run the algorithm (e.g., convergence monitoring routine)
  - information about the current state (e.g., iteration number)
Creating the SLES Context

- **C/C++ version**
  
  ```
  ierr = SLESCreate(MPI_COMM_WORLD,&sles);
  ```

- **Fortran version**
  
  ```
  call SLESCreate(MPI_COMM_WORLD,sles,ierr)
  ```

- Provides an **identical** user interface for all linear solvers
  - uniprocessor and parallel
  - real and complex numbers
Linear Solvers in PETSc 2.0

**Krylov Methods (KSP)**

- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

**Preconditioners (PC)**

- Block Jacobi
- Overlapping Additive Schwarz
- ICC, ILU via BlockSolve95
- ILU(k), LU (sequential only)
- etc.
Basic Linear Solver Code (C/C++)

```c
SLES  sles;  /* linear solver context */
Mat   A;     /* matrix */
Vec   x, b;  /* solution, RHS vectors */
int   n, its;  /* problem dimension, number of iterations */

MatCreate(MPI_COMM_WORLD,n,n,&A);  /* assemble matrix */
VecCreate(MPI_COMM_WORLD,n,&x);
VecDuplicate(x,&b);                        /* assemble RHS vect or */

SLESCreate(MPI_COMM_WORLD,&sles);
SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN);
SLESSetFromOptions(sles);
SLESSolve(sles,b,x,&its);
```

solvers:
linear
beginner
Basic Linear Solver Code (Fortran)

SLES    sles
Mat     A
Vec     x, b
integer n, its, ierr

call  MatCreate(MPI_COMM_WORLD,n,n,A,ierr)
call  VecCreate(MPI_COMM_WORLD,n,x,ierr)
call  VecDuplicate(x,b,ierr)

               C    then assemble matrix and right-hand-side vector

call  SLESCreate(MPI_COMM_WORLD,sles,ierr)
call  SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN,ierr)
call  SLESSetFromOptions(sles,ierr)
call  SLESSolve(sles,b,x,its,ierr)
Setting Solver Options at Runtime

- **-ksp_type** [cg, gmres, bcgs, tfqmr, ...]
- **-pc_type** [lu, ilu, jacobi, sor, asm, ...]

- **-ksp_max_it** <max_iters>
- **-ksp_gmres_restart** <restart>
- **-pc_asm_overlap** <overlap>
- **-pc_asm_type** [basic, restrict, interpolate, none]
- etc ...

1. beginner
2. intermediate
Linear Solvers: Monitoring Convergence

- **-ksp_monitor** - Prints preconditioned residual norm
- **-ksp_xmonitor** - Plots preconditioned residual norm
- **-ksp_truemonitor** - Prints true residual norm \( || b-Ax || \)
- **-ksp_xtruemonitor** - Plots true residual norm \( || b-Ax || \)
- **User-defined monitors, using callbacks**

**solvency:**

1. beginner
2. intermediate
3. advanced
Helmholtz: Scalability

128x512 grid, wave number = 13, IBM SP
GMRES(30)/Restricted Additive Schwarz
1 block per proc, 1-cell overlap, ILU(1) subdomain solver

<table>
<thead>
<tr>
<th>Procs</th>
<th>Iterations</th>
<th>Time (Sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>221</td>
<td>163.01</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>222</td>
<td>81.06</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>224</td>
<td>37.36</td>
<td>4.4</td>
</tr>
<tr>
<td>8</td>
<td>228</td>
<td>19.49</td>
<td>8.4</td>
</tr>
<tr>
<td>16</td>
<td>229</td>
<td>10.85</td>
<td>15.0</td>
</tr>
<tr>
<td>32</td>
<td>230</td>
<td>6.37</td>
<td>25.6</td>
</tr>
</tbody>
</table>
SLES: Review of Basic Usage

• SLESCreate ( ) - Create SLES context
• SLESSetOperators ( ) - Set linear operators
• SLESSetFromOptions ( ) - Set runtime solver options for [SLES, KSP, PC]
• SLESSolve ( ) - Run linear solver
• SLESView ( ) - View solver options actually used at runtime (alternative: -sles_view)
• SLESDestroy ( ) - Destroy solver
# SLES: Review of Selected Preconditioner Options

## Functionality

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set preconditioner type</td>
<td>PCSetType( )</td>
<td>-pc_type [lu,ilu,jacobi, sor,asm,...]</td>
</tr>
<tr>
<td>Set level of fill for ILU</td>
<td>PCILULevels( )</td>
<td>-pc_ilu_levels &lt;levels&gt;</td>
</tr>
<tr>
<td>Set SOR iterations</td>
<td>PCSORSetIteration( )</td>
<td>-pc_sor_its &lt;its&gt;</td>
</tr>
<tr>
<td>Set SOR parameter</td>
<td>PCSORSetOmega( )</td>
<td>-pc_sor_omega &lt;omega&gt;</td>
</tr>
<tr>
<td>Set additive Schwarz variant</td>
<td>PCASMSelType( )</td>
<td>-pc_asm_type [basic, restrict,interpolate,none]</td>
</tr>
<tr>
<td>Set subdomain solver options</td>
<td>PCGetSubSLES( )</td>
<td>-sub_pc_type &lt;pctype&gt; -sub_ksp_type &lt;kspetype&gt; -sub_ksp_rtol &lt;rtol&gt;</td>
</tr>
</tbody>
</table>

And many more options...

**beginner**

**intermediate**
# SLES: Review of Selected Krylov Method Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Krylov method</td>
<td>KSPSetType()</td>
<td>-ksp_type [cg, gmres, bcgs, tfqmr, cgs, ...]</td>
</tr>
<tr>
<td>Set monitoring routine</td>
<td>KSPSetMonitor()</td>
<td>-ksp_monitor, –ksp_xmonitor, -ksp_truemonitor, -ksp_xtruemonitor</td>
</tr>
<tr>
<td>Set convergence tolerances</td>
<td>KSPSetTolerances()</td>
<td>-ksp_rtol &lt;rt&gt; -ksp_atol &lt;at&gt; -ksp_max_its &lt;its&gt;</td>
</tr>
<tr>
<td>Set GMRES restart parameter</td>
<td>KSPGMRESSetRestart()</td>
<td>-ksp_gmres_restart &lt;restart&gt;</td>
</tr>
<tr>
<td>Set orthogonalization routine for GMRES</td>
<td>KSPGMRESSetOrthogonalization()</td>
<td>-ksp_unmodifiedgramschmidt -ksp_irorthog</td>
</tr>
</tbody>
</table>

And many more options...

1 beginner

2 intermediate

solvers: linear: Krylov methods
SLES: Example Programs

Location: petsc/src/sles/examples/tutorials/

- ex1.c, ex1f.F - basic uniprocessor codes
  - ex23.c - basic parallel code
  - ex11.c - using complex numbers

- ex4.c - using different linear system and preconditioner matrices
- ex9.c - repeatedly solving different linear systems
  - ex22.c - 3D Laplacian using multigrid

- ex15.c - setting a user-defined preconditioner

And many more examples ...

beginner intermediate advanced

E - on-line exercise

solvers: linear
Nonlinear Solvers (SNES)

**SNES: Scalable Nonlinear Equations Solvers**

- Application code interface
- Choosing the solver
- Setting algorithmic options
- Viewing the solver
- Determining and monitoring convergence
- Matrix-free solvers
- User-defined customizations

**tutorial outline:**
solvers: nonlinear
Nonlinear PDE Solution

Main Routine

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

PC
KSP

Solve
\( F(u) = 0 \)

PETSc

Application Initialization
Function Evaluation
Jacobian Evaluation
Post-Processing

User code
PETSc code

solvers: nonlinear

beginner
Nonlinear Solvers

Goal: For problems arising from PDEs, support the general solution of \( F(u) = 0 \)

User provides:
- Code to evaluate \( F(u) \)
- Code to evaluate Jacobian of \( F(u) \) (optional)
  - or use sparse finite difference approximation
  - or use automatic differentiation (coming soon!)
Nonlinear Solvers (SNES)

• Newton-based methods, including
  – Line search strategies
  – Trust region approaches
  – Pseudo-transient continuation
  – Matrix-free variants

• User can customize all phases of the solution process
Sample Nonlinear Application: Driven Cavity Problem

- Velocity-vorticity formulation
- Flow driven by lid and/or bouyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- source code:
  
  petsc/src/snes/examples/tutorials/ex8.c

Solution Components

- velocity: $u$
- vorticity: $\zeta$
- temperature: $T$
- velocity: $v$

Application code author: D. E. Keyes
Basic Nonlinear Solver Code (C/C++)

```c
SNES      snes;         /*  nonlinear solver context  */
Mat       J;             /*  Jacobian matrix  */
Vec       x, F;          /*  solution, residual vectors  */
int       n, its;        /*  problem dimension, number of iterations  */
ApplicationCtx  usercontext;   /*  user-defined application context  */

... MatCreate(MPI_COMM_WORLD,n,n,&J);
VecCreate(MPI_COMM_WORLD,n,&x);
VecDuplicate(x,&F);

SNESCreate(MPI_COMM_WORLD,SNES_NONLINEAR_EQUATIONS,&snes);
SNESSetFunction(snes,F,EvaluateFunction,usercontext);
SNESSetJacobian(snes,J,EvaluateJacobian,usercontext);
SNESSetFromOptions(snes);
SNESolve(snes,x,&its);
```

beginner

solvers:
nonlinear
Basic Nonlinear Solver Code (Fortran)

```fortran
SNES   snes
Mat    J
Vec    x, F
int    n, its

...  
call MatCreate(MPI_COMM_WORLD,n,n,J,ierr)
call VecCreate(MPI_COMM_WORLD,n,x,ierr)
call VecDuplicate(x,F,ierr)

call SNESCreate(MPI_COMM_WORLD
&                     SNES_NONLINEAR_EQUATIONS,snes,ierr)
call SNESFunction(snes,F,EvaluateFunction,PETSC_NULL,ierr)
call SNESJacobian(snes,J,EvaluateJacobian,PETSC_NULL,ierr)
call SNESFromOptions(snes,ierr)
call SNESolve(snes,x,its,ierr)
```
Solvers Based on Callbacks

• User provides routines to perform actions that the library requires. For example,
  - \texttt{SNESSetFunction(SNES,...)}
    • \texttt{uservector} - vector to store function values
    • \texttt{userfunction} - name of the user’s function
    • \texttt{usercontext} - pointer to private data for the user’s function

• Now, whenever the library needs to evaluate the user’s nonlinear function, the solver may call the application code directly with its own local state.

• \texttt{usercontext}: serves as an application context object. Data are handled through such opaque objects; the library never sees irrelevant application data

beginner
Uniform access to all linear and nonlinear solvers

- `ksp_type [cg, gmres, bcgs, tfqmr, …]`
- `pc_type [lu, ilu, jacobi, sor, asm, …]`
- `snes_type [ls, tr, …]`

- `snes_line_search <line search method>`
- `sles_ls <parameters>`
- `snes_convergence <tolerance>`
- `etc…`

beginner  intermediate

solvers: nonlinear
SNES: Review of Basic Usage

- **SNESCreate()**: Create SNES context
- **SNESSetFunction()**: Set function eval. routine
- **SNESSetJacobian()**: Set Jacobian eval. routine
- **SNESSetFromOptions()**: Set runtime solver options for [SNES, SLES, KSP, PC]
- **SNESolve()**: Run nonlinear solver
- **SNESView()**: View solver options actually used at runtime (alternative: -snes_view)
- **SNESDestroy()**: Destroy solver

beginner

solvers: nonlinear
## SNES: Review of Selected Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set nonlinear solver</td>
<td>SNESSetType( )</td>
<td>-snes_type [ls,tr,umls,umtr,…]</td>
</tr>
<tr>
<td>Set monitoring routine</td>
<td>SNESSetMonitor( )</td>
<td>-snes_monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>–snes_xmonitor, …</td>
</tr>
<tr>
<td>Set convergence tolerances</td>
<td>SNESSetTolerances( )</td>
<td>-snes_rtol &lt; rt&gt;</td>
</tr>
<tr>
<td>Set line search routine</td>
<td>SNESSetLineSearch( )</td>
<td>-snes_atol &lt; at&gt;</td>
</tr>
<tr>
<td>View solver options</td>
<td>SNESView( )</td>
<td>-snes_max_its &lt; its&gt;</td>
</tr>
<tr>
<td>Set linear solver options</td>
<td>SNESGetSLES( )</td>
<td>-snes_eq_ls [cubic,quadratic,…]</td>
</tr>
<tr>
<td></td>
<td>SLESGetKSP( )</td>
<td>-snes_view</td>
</tr>
<tr>
<td></td>
<td>SLESGetPC( )</td>
<td>-ksp_type &lt; ksptype&gt;</td>
</tr>
<tr>
<td>And many more options...</td>
<td></td>
<td>-ksp_rtol &lt; krt&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-pc_type &lt; pctype&gt;</td>
</tr>
</tbody>
</table>

solvers: nonlinear
SNES: Example Programs

Location: petsc/src/snes/examples/tutorials/

- **ex1.c, ex1f.F** - basic uniprocessor codes
- **ex4.c, ex4f.F** - uniprocessor nonlinear PDE (1 DoF per node)
- **E ex5.c, ex5f.F, ex5f90.F** - parallel nonlinear PDE (1 DoF per node)
- **E ex18.c** - parallel radiative transport problem with multigrid
- **E ex19.c** - parallel driven cavity problem with multigrid

And many more examples ...

beginner intermediate

solvers: nonlinear

- on-line exercise
Timestepping Solvers (TS) (and ODE Integrators)

- Application code interface
- Choosing the solver
- Setting algorithmic options
- Viewing the solver
- Determining and monitoring convergence
- User-defined customizations
Time-Dependent PDE Solution

Main Routine

Timestepping Solvers (TS)

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

PETSc

Solve

\[ U_t = F(U, U_x, U_{xx}) \]

Application Initialization

Function Evaluation

Jacobian Evaluation

Post-Processing

User code

PETSc code

beginner

solvers: timestepping
Timestepping Solvers

**Goal:** Support the (real and pseudo) time evolution of PDE systems

\[ U_t = F(U, U_x, U_{xx}, t) \]

User provides:

- Code to evaluate \( F(U, U_x, U_{xx}, t) \)
- Code to evaluate Jacobian of \( F(U, U_x, U_{xx}, t) \)
  - or use sparse finite difference approximation
  - or use automatic differentiation (coming soon!)
Sample Timestepping Application: Burger’s Equation

\[ U_t = U U_x + \varepsilon U_{xx} \]
\[ U(0,x) = \sin(2\pi x) \]
\[ U(t,0) = U(t,1) \]
Actual Local Function Code

\[ U_t = F(t, U) = U_i \frac{(U_{i+1} - U_{i-1})}{2h} + \varepsilon \frac{(U_{i+1} - 2U_i + U_{i-1})}{h^2} \]

Do 10, i=1,localsize

\[ F(i) = \frac{.5}{h}u(i)(u(i+1) - u(i-1)) + \varepsilon \frac{(u(i+1) - 2.0u(i) + u(i-1))}{h^2} \]

10 continue
Timestepping Solvers

- Euler
- Backward Euler
- Pseudo-transient continuation
- Interface to PVODE, a sophisticated parallel ODE solver package by Hindmarsh et al. of LLNL
  - Adams
  - BDF
Timestepping Solvers

- Allow full access to all of the PETSc
  - nonlinear solvers
  - linear solvers
  - distributed arrays, matrix assembly tools, etc.
- User can customize all phases of the solution process
TS: Review of Basic Usage

- TSCreate() - Create TS context
- TSSetRHSFunction() - Set function eval. routine
- TSSetRHSJacobian() - Set Jacobian eval. routine
- TSSetFromOptions() - Set runtime solver options for [TS, SNES, SLES, KSP, PC]
- TSSolve() - Run timestepping solver
- TSView() - View solver options actually used at runtime (alternative: -ts_view)
- TSDestroy() - Destroy solver
# TS: Review of Selected Options

## Functionality

<table>
<thead>
<tr>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set timestepping solver</strong></td>
<td><strong>-ts_type [euler,beuler,pseudo, ...]</strong></td>
</tr>
<tr>
<td><strong>Set monitoring routine</strong></td>
<td><strong>-ts_monitor</strong></td>
</tr>
<tr>
<td><strong>-ts_xmonitor, ...</strong></td>
<td></td>
</tr>
</tbody>
</table>

## Set timestep duration

<table>
<thead>
<tr>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TSSetDuration( )</strong></td>
<td><strong>-ts_max_steps &lt;maxsteps&gt;</strong></td>
</tr>
<tr>
<td><strong>-ts_max_time &lt;maxtime&gt;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-ts_view</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-snes_monitor -snes_rtol &lt;rt&gt;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-ksp_type &lt;ksptype&gt;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-ksp_rtol &lt;rt&gt;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-pc_type &lt;pctype&gt; ...</strong></td>
<td></td>
</tr>
</tbody>
</table>

And many more options...

beginner  intermediate  

solvers: timestepping
TS: Example Programs

Location: petsc/src/ts/examples/tutorials/

- ex1.c, ex1f.F - basic uniprocessor codes (time-dependent nonlinear PDE)
- ex2.c, ex2f.F - basic parallel codes (time-dependent nonlinear PDE)
- ex3.c - uniprocessor heat equation
- ex4.c - parallel heat equation

And many more examples ...

beginner
intermediate

E - on-line exercise

solvers: timestepping
Mesh Definitions: For Our Purposes

- **Structured**: Determine neighbor relationships purely from logical I, J, K coordinates
- **Semi-Structured**: In well-defined regions, determine neighbor relationships purely from logical I, J, K coordinates
- **Unstructured**: Do not explicitly use logical I, J, K coordinates
Structured Meshes

- PETSc support provided via DA objects
• One is always free to manage the mesh data as if unstructured
• PETSc does not currently have high-level tools for managing such meshes (though lower-level VecScatter utilities provide support)
Semi-Structured Meshes

• No explicit PETSc support
  – OVERTURE-PETSc for composite meshes
  – SAMRAI-PETSc for AMR
Data Layout and Ghost Values: Usage Concepts

Managing **field data layout and required ghost values** is the key to high performance of most PDE-based parallel programs.

**Mesh Types**
- Structured
  - DA objects
- Unstructured
  - VecScatter objects

**Usage Concepts**
- Geometric data
- Data structure creation
- Ghost point updates
- Local numerical computation

**tutorial outline:**
- data layout

**important concepts**
Ghost Values

Ghost values: To evaluate a local function $f(x)$, each process requires its local portion of the vector $x$ as well as its ghost values -- or bordering portions of $x$ that are owned by neighboring processes.
Communication and Physical Discretization

Communication

- Geometric Data
- Data Structure Creation
- Ghost Point Data Structures
- Ghost Point Updates

Local Numerical Computation

- Loops over I,J,K indices

Structure meshes

- stencil [implicit]
- DACreate
- DA AO
- DAGlobalToLocal

Unstructured meshes

- elements
- edges
- vertices
- VecScatterCreate
- VecScatter AO
- VecScatter

Beginner

Intermediate

Data layout
DA: Parallel Data Layout and Ghost Values for Structured Meshes

- Local and global indices
- Local and global vectors
- DA creation
- Ghost point updates
- Viewing
Communication and Physical Discretization: Structured Meshes

**Communication**

- Geometric Data
- Data Structure Creation
- Ghost Point Data Structures
- Ghost Point Updates

**Local Numerical Computation**

- Loops over I,J,K indices

*structured meshes*

- stencil [implicit]
- DACreate( )
- DA AO
- DAGlobalToLocal( )

Data layout: distributed arrays
Global and Local Representations

**Global**: each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

**Local**: each process stores a unique local set of vertices *as well as* ghost nodes from neighboring processes

Data layout: distributed arrays
Logically Regular Meshes

• DA - Distributed Array: object containing information about vector layout across the processes and communication of ghost values

• Form a DA
  – DACreateXX(…..,DA *)

• Update ghostpoints
  – DAGlobalToLocalBegin(DA,…)
  – DAGlobalToLocalEnd(DA,…)

beginner
Distributed Arrays

Data layout and ghost values

Box-type stencil

Star-type stencil

Proc 0 Proc 1

Proc 0 Proc 1

Proc 10

Proc 10
Vectors and DAs

- The DA object contains information about the data layout and ghost values, but **not** the actual field data, which is contained in PETSc vectors.

- **Global vector:** parallel
  - each process stores a unique local portion
  - `DACreateGlobalVector(DA da, Vec *gvec);`

- **Local work vector:** sequential
  - each processor stores its local portion plus ghost values
  - `DACreateLocalVector(DA da, Vec *lvec);`
  - uses “natural” local numbering of indices (0,1,\ldots,nlocal-1)
DACreate1d(...,*DA)

- MPI_Comm - processors containing array
- DA_STENCIL_[BOX,STAR]
- DA_[NONPERIODIC,XPERIODIC]
- number of grid points in x-direction
- degrees of freedom per node
- stencil width
- ...

data layout: distributed arrays
DACreate2d(…,*DA)

- …
- DA_[NON,X,Y,XY]PERIODIC
- number of grid points in x- and y-directions
- processors in x- and y-directions
- degrees of freedom per node
- stencil width
- …

And similarly for DACreate3d()
Updating the Local Representation

Two-step process that enables overlapping computation and communication

- `DAGlobalToLocalBegin(DA,
  - `Vec global_vec,
  - `INSERT_VALUES or ADD_VALUES
  - `Vec local_vec);
- `DAGlobalToLocalEnd(DA,...)
Unstructured Meshes

• Setting up communication patterns is much more complicated than the structured case due to
  – mesh dependence
  – discretization dependence
Sample Differences Among Discretizations

- Cell-centered
- Vertex-centered
- Cell and vertex centered (e.g., staggered grids)
- Mixed triangles and quadrilaterals
Communication and Physical Discretization

**Communication**

- Geometric Data
- Data Structure Creation
- Ghost Point Data Structures
- Ghost Point Updates

**Local Numerical Computation**

- Loops over I,J,K indices

**Structured Mesh**

- stencil [implicit]
- DACreate( )
- DA AO
- DAGlobalToLocal( )

**Unstructured Mesh**

- elements, edges, vertices
- VecScatterCreate( )
- VecScatter AO
- VecScatter( )

**Data Layout**

- beginner
- intermediate
Driven Cavity Model

Example code: petsc/src/snes/examples/tutorials/ex8.c

- Velocity-vorticity formulation, with flow driven by lid and/or bouyancy
- Finite difference discretization with 4 DoF per mesh point

Solution Components

- velocity: \( u \)
- velocity: \( v \)
- vorticity: \( \zeta \)
- temperature: \( T \)

1 beginner 2 intermediate

solvers: nonlinear
Driven Cavity Program

- **Part A**: Parallel data layout
- **Part B**: Nonlinear solver creation, setup, and usage
- **Part C**: Nonlinear function evaluation
  - ghost point updates
  - local function computation
- **Part D**: Jacobian evaluation
  - default colored finite differencing approximation
- Experimentation
Driven Cavity Solution Approach

Main Routine

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

Solve \( F(u) = 0 \)

PETSc

Application Initialization

Function Evaluation

Jacobian Evaluation

Post-Processing

solvers: nonlinear
Driven Cavity:
Running the program (1)

Matrix-free Jacobian approximation with no preconditioning (via -snes_mf) … does not use explicit Jacobian evaluation

- 1 processor: (thermally-driven flow)
  - mpirun -np 1 ex8 -snes_mf -snes_monitor -grashof 1000.0 -lidvelocity 0.0

- 2 processors, view DA (and pausing for mouse input):
  - mpirun -np 2 ex8 -snes_mf -snes_monitor -da_view_draw -draw_pause -1

- View contour plots of converging iterates
  - mpirun ex8 -snes_mf -snes_monitor -snes_vecmonitor

beginner
solvers: nonlinear
Debugging and Error Handling

- Automatic generation of tracebacks
- Detecting memory corruption and leaks
- Optional user-defined error handlers
Sample Error Traceback

Breakdown in ILU factorization due to a zero pivot

```plaintext
[maple] mpirun -np 1 ex1

ex1
  on a solaris named maple by balay Thu Mar 18 15:11:48 1999
  Libraries linked from /home/balay/spetsc/lib/libg_c++/solaris

[0]PETSC ERROR: MatLUFactorSymbolic_SeqAIJ() line 58 in
  src/mat/impls/aij/seq/aijfact.c
[0]PETSC ERROR: Detected zero pivot in factor!
[0]PETSC ERROR: Empty row in matrix
[0]PETSC ERROR: MatLUFactorSymbolic() line 1281 in
  src/mat/interface/matrix.c
[0]PETSC ERROR: PCSetUp_LU() line 170 in src/sles/pc/impls/lu/lu.c
[0]PETSC ERROR: PCSetUp() line 544 in src/sles/pc/interface/precon.c
[0]PETSC ERROR: SLESSetUp() line 383 in src/sles/interface/sles.c
[0]PETSC ERROR: SLESSolve() line 469 in src/sles/interface/sles.c
[0]PETSC ERROR: main() line 142 in src/sles/examples/tutorials/ex1.c
[0] MPI Abort by user Aborting program!
[0] Aborting program!
p0_27843: p4_error: : 1
```

beginner debugging and errors
Sample Memory Corruption Error

```
[mapper] mpirun -np 1 ex2 -trmalloc_off
[mapper] mpirun -np 1 ex2 -trmalloc

ex2
  on a solaris named maple by balay Thu Mar 18 14:04:40 1999
  Libraries linked from /home/balay/spetsc/lib/libc++/solaris

PetcTrFrieDefault called from main() line 20 in
  src/sles/examples/tutorials/ex2.c
Block [id=0(50)] at address 27a958 is corrupted (probably write past end)
Block allocated in main() line 17 in src/sles/examples/tutorials/ex2.c
[0]PETSC ERROR: PetscTrFreeDefault() line 351 in src/sys/src/memory/mtr.c
[0]PETSC ERROR: Memory corruption!
[0]PETSC ERROR: Corrupted memory
[0]Total space allocated 80 bytes
[0]80 bytes main() line 17 in src/sles/examples/tutorials/ex2.c
```

Emacs: ex2.out (Fundamental)--l17--All

beginner

debugging and errors
Sample Out-of-Memory Error

```
[maple] mpirun -np 1 ex3
-------------------------------------------------------------------
ex3
on a solaris named maple by balay Thu Mar 18 14:14:00 1999
Libraries linked from /home/balay/spetsc/lib/libg_c++/solaris
-------------------------------------------------------------------
[0]PETSC ERROR: unknownfunction() line 16 in
    src/sles/examples/tutorials/ex3.c
[0]PETSC ERROR: Out of memory. This could be due to allocating
[0]PETSC ERROR: too large an object or bleeding by not properly
[0]PETSC ERROR: destroying unneeded objects.
[0]PETSC ERROR: Memory allocated 16017800, Memory used 7593984
[0]Total space allocated 16017800 bytes
[ 0] 8000000 bytes main() line 16 in
    src/sles/examples/tutorials/ex3.c
[ 0] 8000000 bytes main() line 16 in
    src/sles/examples/tutorials/ex3.c
[ 0]    24 bytes FListGetPathAndFunction() line 29 in
    src/sys/src/dll/reg.c
[0] MPI Abort by user Aborting program!
[0] Aborting program!
p0_27281: p4_error: : 1
```

beginner debugging and errors
Sample Floating Point Error

```
[maple] mpirun -np 1 ex4 -fp_trap
------------------------
ex4
  on a solaris named maple by balay Thu Mar 18 14:59:30 1999
  Libraries linked from /home/balay/spetsc/lib/libg_c++/solaris
------------------------
[0] PETSC ERROR: unknownfunction() line 0 in unknown file
[0] PETSC ERROR: Signal received!
[0] PETSC ERROR: Caught signal FPE:
[0] PETSC ERROR: Floating Point Exception, probably divide by
[0] PETSC ERROR: zero. Try option -start_in_debugger or
[0] PETSC ERROR: -on_error_attach_debugger to determine where
[0] PETSC ERROR: problem occurs

[0] MPI Abort by user Aborting program!
[0] Aborting program!
p0_27734: p4_error: = 1
```

beginner debugging and errors
Profiling and Performance Tuning

Profiling:
- Integrated profiling using -log_summary
- Profiling by stages of an application
- User-defined events

Performance Tuning:
- Matrix optimizations
- Application optimizations
- Algorithmic tuning
Profiling

• Integrated monitoring of
  – time
  – floating-point performance
  – memory usage
  – communication

• All PETSc events are logged if compiled with -DPETSC_LOG (default); can also profile application code segments

• Print summary data with option: -log_summary

• See supplementary handout with summary data
Conclusion

- Summary
- New features
- Interfacing with other packages
- Extensibility issues
- References
Summary

• Using callbacks to set up the problems for ODE and nonlinear solvers
• Managing data layout and ghost point communication with DAs and VecScatters
• Evaluating parallel functions and Jacobians
• Consistent profiling and error handling
Multigrid Support:
Recently simplified for structured grids

Linear Example:

- 3-dim linear problem on mesh of dimensions $mx \times my \times mz$
  - stencil width = $sw$, degrees of freedom per point = $dof$
  - using piecewise linear interpolation
  - $\text{ComputeRHS()}$ and $\text{ComputeMatrix()}$ are user-provided functions

- $\text{DAMG \; \ast damg;}$
- $\text{DAMGCreate}(\text{comm,nlevels,NULL,} & \text{damg})$
- $\text{DAMGSetGrid}(\text{damg,3,DA_NONPERIODIC,DA_STENCIL_STAR,}\
  \text{mx,my,mz,sw,dof})$
- $\text{DAMGSetSLES}(\text{damg,ComputeRHS,ComputeMatrix})$
- $\text{DAMGSolve}(\text{damg})$
- $\text{solution} = \text{DAMGGetx}(\text{damg})$

All standard SLES, PC and MG options apply.
Multigrid Support

Nonlinear Example:

- 3-dim nonlinear problem on mesh of dimensions \(mx \times my \times mz\)
  - stencil width = \(sw\), degrees of freedom per point = \(dof\)
  - using piecewise linear interpolation
- \(\text{ComputeFunc}()\) and \(\text{ComputeJacobian}()\) are user-provided functions

- \text{DAMG} *damg;
- \text{DAMGCreate}(\text{comm},n\text{levels},\text{NULL},\&\text{damg})
- \text{DAMGSetGrid}(\text{damg},3,DA\_NONPERIODIC,DA\_STENCIL\_STAR, mx,my,mz,sw,dof)
- \text{DAMGSetSNES}(\text{damg},\text{ComputeFunc},\text{ComputeJacobian})
- \text{DAMGSolve}(\text{damg})
- solution = \text{DAMGGetx}(\text{damg})

All standard SNES, SLES, PC and MG options apply.
Using PETSc with Other Packages:

Overture

- Overture is a framework for generating discretizations of PDEs on composite grids.
- PETSc can be used as a “black box” linear equation solver (a nonlinear equation solver is under development).
- Advanced features of PETSc such as the runtime options database, profiling, debugging info, etc., can be exploited through explicit calls to the PETSc API.
Overture Essentials

- Read the grid
  
  ```java
  CompositeGrid cg;
  getFromADatabase(cg,nameOfOGFile);
  cg.update();
  ```

- Create differential operators for the grid
  
  ```java
  int stencilSize = pow(3,cg.numberOfDimensions())+1);
  CompositeGridOperators ops(cg);
  ops.setStencilSize(stencilSize);
  ```

- Create grid functions to hold matrix and vector values
- Attach the operators to the grid functions
- Assign values to the grid functions
- Create an Oges (Overlapping Grid Equation Solver) object to solve the system
Constructing Matrix Coefficients

Laplace operator with Dirichlet BC’s:

- Make a grid function to hold the matrix coefficients:
  ```
  Range all;
  realCompositeGridFunction coeff(cg,stencilSize,all,all,all);
  ```

- Attach operators to this grid function:
  ```
  coeff.setOperators(ops);
  ```

- Designate this grid function for holding matrix coefficients:
  ```
  coeff.setIsACoefficientMatrix(TRUE,stencilSize);
  ```

- Get the coefficients for the Laplace operator:
  ```
  coeff=ops.laplacianCoefficients();
  ```

- Fill in the coefficients for the boundary conditions:
  ```
  coeff.applyBoundaryConditionCoefficients(0,0,dirichlet,allBoundaries);
  ```

- Fill in extrapolation coefficients for defining ghost cells:
  ```
  coeff.applyBoundaryConditionCoefficients(0,0,extrapolate,allBoundaries);
  ```
  ```
  coeff.finishBoundaryConditions();
  ```

software interfacing: Overture
Simple Usage of PETSc through Oges

PETSc API can be hidden from the user

- **Make the solver:**
  
  Oges solver(cg);

- **Set solver parameters:**
  
  solver.set(OgesParameters::THEsolverType,OgesParameters::PETSc);
  solver.set(blockJacobiPreconditioner);
  solver.set(gmres);

- **Solve the system:**
  
  solver.solve(sol,rhs);

- Hides explicit matrix and vector conversions
- Allows easy swapping of solver types (i.e., PETSc, Yale, SLAP, etc.)
Advanced usage of PETSc with Oges

Exposing the PETSc API to the user

- **Set up PETSc:**
  
PetscInitialize(&argc,&argv,...);
PcRegister("MyPC",...);

- **Build a PETScEquationSolver via Oges:**
  
solver.set(OgesParameters::THEsolverType,OgesParameters::PETSc);
solver.buildEquationSolver(solver.parameters.solver.solver);

- **Use Oges for matrix and vector conversions:**
  
solver.formMatrix();
solver.formRhsAndSolutionVectors(sol,rhs);

- **Get a pointer to the PETScEquationSolver:**
  
pes=(PETScEquationSolver*)solver.equationSolver[solver.parameters.solver.solver];

- **Use PETSc API directly:**
  
PCSetType(pes->pc,MyPC);
SLESSolve(pes->sles,pes->xsol,pes->brhs,&its);

- **Use Oges to convert vector into GridFunction:**
  
solver.storeSolutionIntoGridFunction();
#include "Overture.h"
#include "CompositeGridOperators.h"
#include "Oges.h"

int main() {
    printf("This is Overture’s Primer example 7.C");
    // Read in Composite Grid generated by Ogen:
    String nameOfOGFile="TheGrid.hdf";
    CompositeGrid cg;
    getFromADataBase(cg,nameOfOGFile);
    cg.update();

    // Make some differential operators:
    CompositeGridOperators op(cg);
    int stencilSize=pow(3,cg.numberOfDimensions())+1;
    op.setStencilSize(stencilSize);

    // Make grid functions to hold vector coefficients:
    realCompositeGridFunction u(cg),f(cg);

    // Assign the right hand side vector coefficients:
    ...

    // Make a grid function to hold the matrix coefficients:
    Range all;
    realCompositeGridFunction coeff(cg,stencilSize,all,all,all);
    // Attach operators to this grid function:
    coeff.setOperators(op);
    // Designate this grid function for holding matrix coefficients:
    coeff.setIsACoefficientMatrix(TRUE,stencilSize);

    // Get the coefficients for the Laplace operator:
    coeff=op.laplacianCoefficients();
    // Fill in the coefficients for the boundary conditions:
    coeff.applyBoundaryConditionCoefficients(0,0,
        BCTypes::dirichlet,BCTypes::allBoundaries);
    coeff.applyBoundaryConditionCoefficients(0,0,
        BCTypes::extrapolate,BCTypes::allBoundaries);
    coeff.finishBoundaryConditions();

    // Create an Overlapping Grid Equation Solver:
    Oges solver(cg);
    // Tell Oges to use PETSc:
    solver.set(OgesParameters::THEsolverType,
        OgesParameters::PETSc);
    // Tell Oges which preconditioner and Krylov solver to use:
    solver.set(blockJacobiPreconditioner);
    solver.set(gmres);
    // Prescribe the location of the matrix coefficients:
    solver.setCoefficientArray( coeff );
    // Solve the system:
    solver.solve( u,f );

    // Display the solution using Overture’s ASCII format:
    u.display();
    return(0);
}
Advanced PETSc Usage In Overture

```c
#include "mpi.h"
#include "Overture.h"
#include "CompositeGridOperators.h"
#include "Oges.h"
#include "petscpc.h"

EXTERN_C_BEGIN
extern int CreateMyPC(PC);
EXTERN_C_END

char help="This is Overture's Primer example7.C using \n advanced PETSc features. \n Use of the Preconditioner \n 'MyPC' is enabled via the option \n 	–pc_type MyPC";

int main(int argc,char *argv[]) {
  int ierr = PetscInitialize(&argc,&argv,0,help);
  {
    // Allow PETSc to select a Preconditioner I wrote:
    ierr = PCRegister("MyPC",0,"CreateMyPC",CreateMyPC);
    // Read in Composite Grid generated by Ogen:
    String nameOfOGFile="TheGrid.hdf";
    // Determine file with runtime option -file
    PetscTruth flag;
    ierr = OptionsGetString(0,"-file",(char*)nameOfOGFile,&flag); CHKERRA(ierr);
    CompositeGrid cg;
    getFromADataBase(cg,nameOfOGFile);
    cg.update();

    // Make some differential operators: …
    // Make grid functions to hold vector coefficients: …
    // Make a grid function to hold the matrix coefficients: …
    // Create an Overlapping Grid Equation Solver:
    Oges solver(cg);
    // Prescribe the location of the matrix coefficients:
    solver.setCoefficientArray(coeff);
    // Tell Oges to use PETSc:
    solver.set(OgesParameters::THEsolverType,
               OgesParameters::PETSc);
    // Tell Oges which preconditioner and Krylov solver to use:
    solver.set(blockJacobiPreconditioner);
    solver.set(gmres);
    // Allow command line arguments to supercede the above,
    // enabling use of the runtime option: -pc_type MyPC
    solver.setCommandLineArguments(argc,argv);
    // Solve the system:
    solver.solve( u,f );

    // Access PETSc Data Structures:
    PETScEquationSolver &pes = *(PETScEquationSolver *)
    solver.equationSolver[OgesParameters::PETSc];
    // View the actual (PETSc) matrix generated by Overture:
    ierr = MatView(pes.Amx,VIEWER_STDOUT_SELF);
    CHKERRA(ierr);
    // Display the solution using Overture’s ASCII format:
    u.display();
  }
  PetscFinalize();
  return(0);
}
```

software interfacing: Overture
Using PETSc with Other Packages

**ILUDTP - Drop Tolerance ILU**

- Use PETSc **SeqAIJ** or **MPIAIJ** (for block Jacobi or ASM) matrix formats
- `-pc_ilu_use_drop_tolerance <dt,dtcol,maxrowcount>`
  - `dt` – drop tolerance
  - `dtcol` - tolerance for column pivot
  - `maxrowcount` - maximum number of nonzeros kept per row
Using PETSc with Other Packages

**ParMETIS – Graph Partitioning**

- Use PETSc MPIAIJ or MPIAdj matrix formats
- MatPartitioningCreate(MPI_Comm, MatPartitioning ctx)
- MatPartitioningSetAdjacency(ctx, matrix)
- Optional – MatPartitioningSetVertexWeights(ctx, weights)
- MatPartitioningSetFromOptions(ctx)
- MatPartitioningApply(ctx, IS *partitioning)
Using PETSc with Other Packages

**PVODE** – ODE Integrator

- TSCreate(MPI_Comm, TS_NONLINEAR, &ts)
- TSSetType(ts, TS_PVODE)
- ….. regular TS functions
- TSPVODESetType(ts, PVODE_ADAMS)
- ….. other PVODE options
- TSSetFromOptions(ts) – accepts PVODE options
Using PETSc with Other Packages

**SPAI** – Sparse Approximate Inverse

- `PCSetType(pc,PCSPAI)`
- `PCSPAISetXXX(pc,...)` … set SPAI options
- `PCSetFromOptions(pc)` … accepts SPAI options
Using PETSc with Other Packages

Matlab

- PetscMatlabEngineCreate(MPI_Comm, machinename, PetscMatlabEngine eng)
- PetscMatlabEnginePut(eng, PetscObject obj)
  - Vector
  - Matrix
- PetscMatlabEngineEvaluate(eng, ”R = QR(A);”)
- PetscMatlabEngineGet(eng, PetscObject obj)
Using PETSc with Other Packages

SAMRAI

• SAMRAI provides an infrastructure for solving PDEs using adaptive mesh refinement with structured grids.
• SAMRAI developers wrote a new class of PETSc vectors that uses SAMRAI data structures and methods.
• This enables use of the PETSc matrix-free linear and nonlinear solvers.
Sample Usage of SAMRAI with PETSc

Exposes PETSc API to the user

- Make a SAMRAI Vector:
  Samrai_Vector = new SAMRAIVectorReal2<double>(...);
- Generate vector coefficients using SAMRAI
- Create the PETSc Vector object wrapper for the SAMRAI Vector:
  Vec PETSc_Vector = createPETScVector(Samrai_Vector);
- Use PETSc API to solve the system:
  SNESCreate(...);
  SNESSolve(...);

Both PETSc_Vector and Samrai_Vector refer to the same data
Using PETSc with Other Packages

**TAO**

- The Toolkit for Advanced Optimization (TAO) provides software for large-scale optimization problems, including
  - unconstrained optimization
  - bound constrained optimization
  - nonlinear least squares
  - nonlinearly constrained optimization
- TAO uses abstractions for vectors, matrices, linear solvers, etc.; currently PETSc provides these implementations.
- TAO interface is similar to that of PETSc
TAO Interface

TAO_SOLVER tao; /* optimization solver */
Vec x, g; /* solution and gradient vectors */
ApplicationCtx usercontext; /* user-defined context */

TaoInitialize();

/* Initialize Application -- Create variable and gradient vectors x and g */ ... 

TaoCreate(MPI_COMM_WORLD,"tao_lmvm",&tao);
TaoSetFunctionGradient(tao,x,g,FctGrad,(void*)&usercontext);

TaoSolve(tao);

/* Finalize application -- Destroy vectors x and g */ ... 

TaoDestroy(tao);
TaoFinalize();

Similar Fortran interface, e.g., call TaoCreate(...)
Extensibility Issues

• Most PETSc objects are designed to allow one to “drop in” a new implementation with a new set of data structures (similar to implementing a new class in C++).

• Heavily commented example codes include
  – Krylov methods: petsc/src/sles/ksp/impls/cg
  – Preconditioners: petsc/src/sles/pc/impls/jacobi

• Feel free to discuss more details with us in person.
Caveats Revisited

- Developing parallel, non-trivial 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.
- Users are invited to interact directly with us regarding correctness or performance issues by writing to petsc-maint@mcs.anl.gov.
References

- http://www.mcs.anl.gov/petsc/docs
- **Example codes** – docs/exercises/main.htm
- http://www.mpi-forum.org
- **Using MPI (2nd Edition)**, Gropp, Lusk, and Skjellum
- **Domain Decomposition**, Smith, Bjorstad, and Gropp