PETSc Tutorial
Numerical Software Libraries for the Scalable Solution of PDEs
Satish Balay, Kris Buschelman, Bill Gropp, Dinesh Kaushik, Matt Knepley, Lois Curfman McInnes, Barry Smith, Hong Zhang
Mathematics and Computer Science Division
Argonne National Laboratory
http://www.mcs.anl.gov/petsc
Intended for use with version 2.1.0 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
• Introduce PETSc interfaces to other software packages
• Explain 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
  – Free for everyone, including industrial users
  – 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 SPP, IBM SGI Sun
• PCs running Linux or Windows
• PETSc history
  – Begun in September 1991
  – Now: over 8,500 downloads since 1995 (versions 2.0 and 2.1)
• PETSc funding and support
  – National Science Foundation, Multidisciplinary Challenge Program, CISE

The PETSc Team
Satish Balay
Kris Buschelman
Bill Gropp
Dinesh Kaushik
Matt Knepley
Lois Curfman McInnes
Barry Smith
Hong Zhang

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
- motivating examples
- programming paradigm

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

Viewers
- object information visualization

Solvers
- Linear
- Profiling and performance tuning

Solvers (cont.)
- nonlinear
timestepping (and ODEs)

Data layout and ghost values
- structured and unstructured mesh problems

Putting it all together
- a complete example

Debugging and error handling

New features
Using PETSc with other software packages

Tutorial Topics: Using PETSc with Other Packages

- Linear solvers
  - AMG
  - BlockSolve95
  - ILUTP
  - LUSOL
  - SPAI
  - SuperLU

- Optimization software
  - TAO
  - Veltisto

Mesh and discretization tools
- Overture
- SAMRAI
- SUMAAM
- ODE solvers
- PVODE
- Others

- Matlab
- ParMETIS

CFD on an Unstructured Mesh

- 3D incompressible Euler
- Tetrahedral grid
- Up to 11 million unknowns
- Based on a legacy NASA code, FUN3d, developed by W. K. Anderson
- Fully implicit steady-state
- Primary PETSc tools: nonlinear solvers (SNES) and vector scatters (VecScatter)

Results courtesy of Dinesh Kaushik and David Keyes, Old Dominion Univ., partially funded by NSF and ASCI level 2 grant

Fixed-size Parallel Scaling Results (GFlop/s)

Dimension=11,047,096

Fixed-size Parallel Scaling Results (Time in seconds)

Inside the Parallel Scaling Results on ASCI Red

CINDRA M6 wing test case, tetrahedral grid of 2.0 million vertices (about 11 million unknowns) on up to 3072 ASCI Red nodes (each with dual Pentium Pro 333 MHz processors)
Multiphase Flow

- Oil reservoir simulation: fully implicit, time-dependent
- First fully implicit, parallel compositional simulator
- 3D EOS model (8 DoF per cell)
- Structured Cartesian mesh
- Over 4 million cell blocks, 32 million DoF
- Primary PETSc tools: linear solvers (SLES)
  - restarted GMRES with Block-Jacobi preconditioning
  - Point-block ILU(0) on each processor
- Over 10.6 gigaflops sustained performance on 128 nodes of an IBM SP, 90%+ percent parallel efficiency

Results courtesy of collaborators Peng Wang and Jason Abate, Univ. of Texas at Austin, partially funded by DOE ER FE/MICS

PC and SP Comparison

<table>
<thead>
<tr>
<th>Number Processors</th>
<th>PC</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>800</td>
</tr>
<tr>
<td>4</td>
<td>800</td>
<td>1600</td>
</tr>
<tr>
<td>8</td>
<td>1600</td>
<td>3200</td>
</tr>
<tr>
<td>16</td>
<td>3200</td>
<td>6400</td>
</tr>
</tbody>
</table>

179,000 unknowns (22,375 cell blocks)

- PC: Fast ethernet (100 Megabits/second) network of 300 MHz Pentium PCs with 66 MHz bus
- SP: 128 node IBM SP with 160 MHz Power2super processors and 2 memory cards

Structures Simulations

- ALE3D (LLNL structures code) test problems
- Simulation with over 16 million degrees of freedom
- Run on NERSC 512 processor T3E and LLNL ASCI Blue Pacific
- Primary PETSc tools: multigrid linear solvers (SLES)

Results courtesy of Mark Adams (Univ. of California, Berkeley)

ALE3D Test Problem Performance

NERSC Cray T3E Scaled Performance
15,000 DoF per processor

Tutorial Approach

From the perspective of an application programmer:

- Beginner
  - basic functionality, intended for use by most programmers
  - Emphasis of this tutorial: beginner
- Intermediate
  - selecting options, performance evaluation and tuning
  - Emphasis of this tutorial: intermediate
- Advanced
  - user-defined customization of algorithms and data structures
  - Emphasis of this tutorial: advanced
- Developer
  - advanced customizations, intended primarily for use by library developers
  - Emphasis of this tutorial: 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

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>Data Search</td>
<td>Time-Dependent</td>
</tr>
<tr>
<td>PINNED</td>
<td>Time-Marching</td>
</tr>
</tbody>
</table>

Krylov Subspace Methods

<table>
<thead>
<tr>
<th>Preconditioners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
</tr>
<tr>
<td>Damping</td>
</tr>
<tr>
<td>Unsmooth</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

Matrices

<table>
<thead>
<tr>
<th>Distributed Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Sets</td>
</tr>
<tr>
<td>Sparse Row (AIJ)</td>
</tr>
<tr>
<td>Blocked Sparse Row (BAIJ)</td>
</tr>
<tr>
<td>Block Diagonal (BDIAG)</td>
</tr>
<tr>
<td>Dense</td>
</tr>
<tr>
<td>Matrix-free</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

PETSc PDE Application Codes

- CDE Integrators
- Visualization
- Nonlinear Solvers
- Unconstrained Minimization
- Linear Solvers
- Preconditioners = Krylov Methods
- Object-Oriented
- Matrices, Vectors, Indices
- Old Management
- Computing Interface
- Computation and Communication Kernels

What is not in PETSc?

- Discretizations
- Unstructured mesh generation and refinement tools
- Load balancing tools
- Sophisticated visualization capabilities

But PETSc does interface to external software that provides some of this functionality.
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

PETSc emphasis

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

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

```c
#include "petsc.h"
int main( int argc, char *argv[] )
{
  PetscInitialize(&argc,&argv,PETSC_NULL,PETSC_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
```

Fancier Hello World

```c
#include "petsc.h"
int main( int argc, char *argv[] )
{
  int rank;
  PetscInitialize(&argc,&argv,PETSC_NULL,PETSC_NULL);
  MPI_Comm_rank(PETSC_COMM_WORLD,&rank);
  PetscSynchronizedPrintf(PETSC_COMM_WORLD,"Hello World from %d\n",rank);
  PetscSynchronizedFlush(PETSC_COMM_WORLD);
  PetscFinalize();
  return 0;
}
```

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

- What are PETSc vectors?
  - Fundamental objects for storing field solutions, right-hand sides, etc.
  - Each process locally owns a subvector of contiguously numbered global indices
- Create vectors via
  - VecCreate(MPI_Comm, Vec*)
  - MPI_Comm - processors that share the vector
  - number of elements local to this processor
  - or total number of elements
  - VecSetType(Vec, VecType)
    - Where VecType is
      - VEC_SEQ, VEC_MPI, or VEC_SHARED
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 \cdot x )</td>
</tr>
<tr>
<td>VecAXPY(Scalar *a, Vec x, Vec y)</td>
<td>( y = y + a \cdot x )</td>
</tr>
<tr>
<td>VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)</td>
<td>( w = a \cdot x + y )</td>
</tr>
<tr>
<td>VecScale(Scalar *a, Vec x)</td>
<td>( x = a \cdot 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 \cdot y_j )</td>
</tr>
<tr>
<td>VecMax(Vec x, int *idx, double *r)</td>
<td>( r = \max 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 = |x| )</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

Matrices

- What are PETSc matrices?
  - Fundamental objects for storing linear operators (e.g., Jacobians)
- Create matrices via
  - **MatCreate( )**
  - **MPI_Comm** - processors that share the matrix
  - number of local/global rows and columns
- **MatSetType(Mat, MatType)**
  - where MatType is one of
    - default sparse AIJ: MPIAIJ, SEQAIJ
    - block sparse AIJ (for multi-component PETEs): MPIAIJ, SEQAIJ
    - symmetric block sparse AIJ: MPISBAIJ, SSEQSBAIJ
    - block diagonal: MPIDIG, SEQIDIG
    - dense: MPIDENSE, SEQDENSE
    - matrix-free
    - etc.

Matrices and Polymorphism

- Single user interface, e.g.,
  - **MatSetValues()**
  - **Matrix-vector multiplication**
    - **MatMult()**
  - **Matrix viewing**
    - **MatView()**
- Multiple underlying implementations
  - AIJ, block AIJ, symmetric block AIJ, block diagonal, dense, matrix-free, etc.
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)

Matrix Assembly Example

simple 3-point stencil for 1D discretization

```c
Mat   A;
int    column[3], i, start, end;
double value[3];
/* mesh interior */
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
for (i=start; i<end; i++) {
    column[0] = i-1; column[1] = i; column[2] = i+1;
    MatSetValues(A,1,&i,3,column,value,INSERT_VALUES);
}
/* also must set boundary points */
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

Parallel Matrix Distribution

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

```
proc 0
proc 2
proc 3: locally owned rows
proc 4
```

Parallel Matrix Distribution

```
MatGetOwnershipRange(Mat A, int *rstart, int *rend)
- rstart: first locally owned row of global matrix
- rend -1: last locally owned row of global matrix
```

Blocking: Performance Benefits

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

```
Basic
```

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, PetscViewer v);
• Default viewers
  - ASCII (sequential): PETSC_VIEWER_STDOUT_SELF
  - ASCII (parallel): PETSC_VIEWER_STDOUT_WORLD
  - X-windows: PETSC_VIEWER_DRAW_WORLD
• Default ASCII formats
  - PETSC_VIEWER_ASCII_DEFAULT
  - PETSC_VIEWER_ASCII_MATLAB
  - PETSC_VIEWER_ASCII_COMMON
  - PETSC_VIEWER_ASCII_INFO
  - etc.

Viewing Matrix Data

• MatView(Mat A, PetscViewer 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

Solution Components

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

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

Sample Linear Application:
Exterior Helmholtz Problem

Solution Components

Real

Imaginary

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

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)

Linear Solvers (SLES)

- 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

Creating the SLES Context

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

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

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

C then assemble matrix and right-hand-side vector

call SLESSetOperators(sles,A,A,Different_NONZERO_PATTERN,ierr)
call SLESSetFromOptions(sles,ierr)
call SLESSolve(sles,b,x,its,ierr)
call SLESDestroy(sles,ierr)

Customization Options

• Procedural Interface
  – Provides a great deal of control on a usage-by-usage
    basis inside a single code
  – Gives full flexibility inside an application

• Command Line Interface
  – Applies same rule to all queries via a database
  – Enables the user to have complete control at runtime,
    with no extra coding

Setting Solver Options within Code

• SLESGetKSP(SLES sles,KSP *ksp)
  – KSPSetType(KSP ksp,KSPType type)
  – KSPSetTolerances(KSP ksp,PetscReal rtol,
      PetscReal atol,PetscReal dtol, int maxits)
  – etc....

• SLESGetPC(SLES sles,PC *pc)
  – PCSetType(PC pc,PCType)
  – PCASMSetOverlap(PC pc,int overlap)
  – etc....

Recursion: Specifying Solvers for
Schwarz Preconditioner Blocks

• Specify SLES solvers and options with “-sub” prefix, e.g.,
  – Full or incomplete factorization
    -sub_pc_type lu
    -sub_pc_type ilu
    -sub_pc_type <levels>
  – Can also use inner Krylov iterations, e.g.,
    -sub_ksp_type gmres
    -sub_ksp_rtol <rtol>
    -sub_ksp_max_it <maxit>

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 ...

Linear Solvers: Monitoring Convergence

• -ksp_monitor - Prints preconditioned residual norm
• -ksp_xmonitor - Plots preconditioned residual norm
• -ksp_true monitor - Prints true residual norm ||b-Ax||
• -ksp_xtrue monitor - Plots true residual norm ||b-Ax||

• User-defined monitors, using callbacks
Helmholtz: Scalability

128x512 grid, wave number = 13, IBM SP
GMRES(30)/Restricted Additive Schwarz

<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>

1 block per proc, 1-cell overlap, ILU(1) subdomain solver

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
- SLSView() - View solver options actually used at runtime (alternative: -sles_view)
- SLESDestroy() - Destroy solver

SLES: Review of Selected Preconditioner Options

<table>
<thead>
<tr>
<th>Functionality</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>PCILUSetLevels()</td>
<td>-pc_ilu_levels &lt;levels&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>PCASMSetType()</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;ksptype&gt; -sub_ksp_rtol &lt;rtol&gt;</td>
</tr>
</tbody>
</table>

And many more options...

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_showmonitor</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>-kap_orthog</td>
</tr>
</tbody>
</table>

And many more options...

SLES: Runtime Script Example

Viewing SLES Runtime Options
SLES: Example Programs

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

- `ex1.c`, `ex1.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 ...

Profiling and Performance Tuning

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

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

User-defined Events

```c
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT, "User event name", "eventColor");
PetcLogEventBegin(USER_EVENT, 0, 0, 0, 0);
[code to monitor]
PetcLogFlops(user_event_flops);
PetcLogEventEnd(USER_EVENT, 0, 0, 0, 0);
```

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

Nonlinear PDE Solution
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
    - AD support via collaboration with P. Hovland and B. Norris
    - Coming in next PETSc release via automated interface to ADIFOR and ADIC (see [http://www.mcs.anl.gov/autodiff](http://www.mcs.anl.gov/autodiff))

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 buoyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- Source code:
  - petsc/src/snes/examples/tutorials/ex19.c

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,ierr);
VecCreate(MPI_COMM_WORLD,n,x,ierr);
VecDuplicate(x,F,ierr);
SNESCreate(MPI_COMM_WORLD,SNES_NONLINEAR_EQUATIONS,&snes,ierr);
SNESSetFunction(snes,F, EvaluateFunction, PETSC_NULL,ierr);
SNESSetJacobian(snes,J, EvaluateJacobian, PETSC_NULL,ierr);
SNESSetFromOptions(snes,ierr);
SNESSolve(snes,x,&its,ierr);
SNESDestroy(snes,ierr);
```

Basic Nonlinear Solver Code (Fortran)

```fortran
SNES snes
Mat J
Vec x, F
int n, its
...
MatCreate(MPI_COMM_WORLD,n,n,J,ierr)
VecCreate(MPI_COMM_WORLD,n,x,ierr)
VecDuplicate(x,F,ierr)
SNESCreate(MPI_COMM_WORLD,SNES_NONLINEAR_EQUATIONS,&snes,ierr)
SNESSetFunction(snes,F, EvaluateFunction,PETSC_NULL,ierr)
SNESSetJacobian(snes,J, EvaluateJacobian,PETSC_NULL,ierr)
SNESSetFromOptions(snes,ierr)
SNESSolve(snes,x,its,ierr)
SNESDestroy(snes,ierr)
```

Solvers Based on Callbacks

- User provides routines to perform actions that the library requires. For example,
  - SNESSetFunction(SNES,_,
  - UserVector - vector to store function values
  - UserFunction - name of the user's function
  - 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.
- UserContext: serves as an application context object. Data are handled through such opaque objects; the library never sees irrelevant application data.
Sample Application Context: Driven Cavity Problem

typedef struct {
  /* ---------------- basic application data ---------------- */
  double         lid_velocity, prandtl, grashof; /* problem parameters */
  int            mx, my; /* discretization parameters */
  int            mc; /* number of DoF per node */
  int            draw_contours; /* flag - drawing contours */
  /* ----------------- parallel data ---------------------- */
  MPI_Comm       comm; /* communicator */
  DA             da; /* distributed array */
  Vec            localF, localX; /* local ghosted vectors */
} AppCtx;

Sample Function Evaluation Code: Driven Cavity Problem

UserComputeFunction(SNES snes, Vec X, Vec F, void *ptr)
{
  AppCtx *user = (AppCtx *) ptr; /* user-defined application context */
  int            istart, iend, jstart, jend; /* local starting and ending grid points */
  Scalar         *f; /* local vector data */
  ...
  /* Communicate nonlocal ghost point data */
  VecGetArray(F, &f);
  /* Compute local function components; insert into f[] */
  VecRestoreArray(F, &f);
  ...
  return 0;
}

Sample Local Computational Loops: Driven Cavity Problem

#define U(i) 4*(i)
#define V(i) 4*(i)+1
#define Omega(i) 4*(i)+2
#define Temp(i) 4*(i)+3
...
for ( j = jstart;  j<jend;  j++ )  {
  row = (j - gys) * gxm + istart - gxs - 1;
  for ( i = istart; i<iend; i++ )  {
    row++;    u = x[U(row)];
    uxx = (two * u - x[U(row-1)] - x[U(row+1)]) / hxdx;
    uyy = (two * u - x[U(row-gxm)] - x[U(row+gxm)]) / hxdy;
    f[U(row)] = uxx + uyy - p5 * (x[Omega(row+gxm)] - x[Omega(row-gxm)]) / hc;
  }
}...

Finite Difference Jacobian Computations

- Compute and explicitly store Jacobian via 1st-order FD
  - Dense: -snes_fd, SNESDefaultComputeJacobian()
  - Sparse via colorings: MatFDColoringCreate(),
    SNESDefaultComputeJacobianColor()
- Matrix-free Newton-Krylov via 1st-order FD, no
  preconditioning unless specifically set by user
  - -snes_mf
- Matrix-free Newton-Krylov via 1st-order FD, user-defined
  preconditioning matrix
  - -snes_mf_operator

Uniform access to all linear and nonlinear solvers

- ksp_type [cg, gmres, bcs, 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....

SNES: Review of Basic Usage

- SNESCreate() - Create SNES context
- SNESFunction() - Set function eval. routine
- SNESSetJacobi() - 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
SNES: Review of Selected Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Routine 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_amonitor, …</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_max_it &lt;its&gt;</td>
</tr>
<tr>
<td>View solver options</td>
<td>SNESSetView( )</td>
<td>-snes_view</td>
</tr>
<tr>
<td>Set linear solver options</td>
<td>SNESSetSLES( )</td>
<td>-snes_ksp_type &lt;ksp_type&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-snes_pc_type &lt;pc_type&gt;</td>
</tr>
</tbody>
</table>

And many more options...

SNES: Example Programs

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

- `ex1.c, ex1.f`: basic uniprocessor codes
- `ex2.c, ex2.f`: uniprocessor nonlinear PDE (1 DoF per node)
- `ex5.c, ex5.f, ex5r0.f`: parallel nonlinear PDE (1 DoF per node)
- `ex18.c`: parallel radiative transport problem with multigrid
- `ex19.c`: parallel driven cavity problem with multigrid

And many more examples...

Timstepping Solvers (TS)

(also ODE Integrators)

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

Sample Timstepping Application: Burger's Equation

\[
\frac{U}{t} = U_x + \varepsilon \ U_{xx}
\]

\[
U(0, x) = \sin(2\pi x)
\]

\[
U(t, 0) = U(t, 1)
\]
**Actual Local Function Code**

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

Do 10, i=1,localsize
\[ F(i) = \frac{(\varepsilon)}{h^2} (U_{i+1} - 2.0U_i + U_{i-1}) + \]
\[ 10 \text{ continue} \]

**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**
- TSSetType( ) -ts_type [euler,beuler,pseudo,…]
- TSSetMonitor() -ts_monitor -ts_xmonitor, …
- TSSetDuration ( ) -ts_max_steps <maxsteps>
  -ts_max_time <maxtime>
- TSView( ) -ts_view
- TSGetSNES( )
  -snes_monitor -snes_rtol <rt>
- SNESGetSLES( )
  -ksp_type <ksptype>
- SLESGetKSP( )
  -ksp_rtol <rt>
- PCGetKSP( ) -pc_type <pctype> …

**TS: Review of Selected Options**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set...</td>
<td>TSSetType( )</td>
<td>-ts_type euler,beuler,pseudo,…</td>
</tr>
<tr>
<td>Set...</td>
<td>TSSetMonitor( )</td>
<td>-ts_monitor -ts_xmonitor, …</td>
</tr>
</tbody>
</table>
| Set...                 | TSSetDuration ( )    | -ts_max_steps <maxsteps>
  -ts_max_time <maxtime>
| View...                | TSView( )            | -ts_monitor -ts_xmonitor, … |
| View...                | SNESGetSLES( )       | -snes_type <slepctype>
  -slep_rtol <rt>
| View...                | SLESGetKSP( )        | -ksp_type <kspctype> |
| View...                | PCGetKSP( )          | -pc_type <pctype> |

**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...
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

Unstructured Meshes

- 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

Parallel 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.

<table>
<thead>
<tr>
<th>Mesh Types</th>
<th>Usage Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured</td>
<td>- Geometric data</td>
</tr>
<tr>
<td>Unstructured</td>
<td>- Data structure creation</td>
</tr>
<tr>
<td></td>
<td>- Ghost point updates</td>
</tr>
<tr>
<td></td>
<td>- Local numerical computation</td>
</tr>
</tbody>
</table>

Ghost Values

- Local node
- Ghost node

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
- stencil (implicit)
- VecScatter
- VecScatterGather

### 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

### Global and Local Representations
- Global Representation:
  - Each process stores a unique local set of vertices (and each vertex is owned by exactly one process)
- Local Representation:
  - Each process stores a unique local set of vertices as well as ghost nodes from neighboring processes

### 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,....)

**tutorial outline:**
- data layout: distributed arrays

---

- Images of slides from an ACTS Toolkit Workshop presentation, focusing on communication and physical discretization, data structures, and structured meshes.
Distributed Arrays

Data layout and ghost values

Box-type stencil

Star-type stencil

Distributed Arrays

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,...,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
- ...

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);
- DAGlobalToLocal End(DA,…)

Ghost Point Scatters:

Burger's Equation Example

C  Do local computations (here u and f are local vectors)
Do 10, i=1,localsize
  f(i) = (.5h)*u(i)*(u(i+1)-u(i-1)) +
       (e/(h*h))*((u(i+1) - 2.0*u(i) + u(i-1))
  continue
Call DALocalToGlobal(da,f,INSERT_VALUES,f_global,ierr)
Unstructured Meshes

- Setting up communication patterns is much more complicated than the structured case due to
  - mesh dependence
  - discretization dependence
    - cell-centered
    - vertex-centered
    - cell and vertex centered (e.g., staggered grids)
    - mixed triangles and quadrilaterals
- Can use VecScatter
  - See additional tutorial material available via PETSc web site

Driven Cavity Model

Example code: petsc/src/snes/examples/tutorials/ex19.c

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

\[ \begin{bmatrix} u, v, \zeta, T \end{bmatrix} \]

Solution Components

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

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

Matrix-free Jacobian approximation with no preconditioning (via \texttt{-snes_mf}) ... does not use explicit Jacobian evaluation

- 1 processor: (thermally-driven flow)
  - mpirun -np 1 ex19 \texttt{-snes_mf -snes_monitor -grashof 1000.0 -lidvelocity 0.0}
- 2 processors, view \texttt{DA} (and pausing for mouse input):
  - mpirun -np 2 ex19 \texttt{-snes_mf -snes_monitor -da_view_draw -draw_pause -1}
- View contour plots of converging iterates
  - mpirun ex19 \texttt{-snes_mf -snes_monitor -snes_vecmonitor}
Debugging and Error Handling

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

Tutorial outline:
- Debugging and errors

Beginner

Debugging
Support for parallel debugging
- `start_in_debugger [gdb,dbx,noxterm]`
- `on_error_attach_debugger [gdb,dbx,noxterm]`
- `on_error_abort`
- `debugger_nodes 0,1`
- `display machinename:0.0`

When debugging, it often useful to place a breakpoint in the function `PetscError()`.  

Sample Error Traceback
Breakdown in ILU factorization due to a zero pivot

Sample Memory Corruption Error

Sample Out-of-Memory Error

Sample Floating Point Error
Conclusion

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

Summary

- Creating data objects
- Setting algorithmic options for linear, nonlinear and ODE solvers
- Using callbacks to set up the problems for nonlinear and ODE solvers
- Managing data layout and ghost point communication
- Evaluating parallel functions and Jacobians
- Consistent profiling and error handling

New Features

- Version 2.1.0
  - Simple interface for multigrid on structured meshes
  - VecPack – manages treating several distinct vectors as one
  - useful for design optimization problems written as a nonlinear system
- Next release
  - Automatically generated Jacobians via ADIC and ADIFOR
    - Fully automated for structured mesh parallel programs using DAs
    - General parallel case under development
- Under development
  - Parallel interface to SuperLU
  - Interface to SLEPc eigenvalue software under development by V. Hernandez and J. Roman
  - Support for CCA-compliant components (see http://www.cca-forum.org)

Multigrid Structured Mesh Support:

DMMG: New Simple Interface

- General multigrid support
  - PC framework wraps MG for use as pre-conditioner
    - See MGSetXXX(), MGGetXXX()
    - can access via -pc_type mg
- User provides coarse grid solver, smoothers, and interpolation/restriction operators
- DMMG - simple MG interface for structured meshes
  - User provides
    - “Local” function evaluation
    - (Optional) local Jacobian evaluation

Multigrid Structured Mesh Support:
Sample Function Computation

```c
int Function(DALocalInfo *info, double **u, double ***f, AppCtx *user)
...
lambda = user->param;
hx = 1.0/(info->mx-1);
hy = 1.0/(info->my-1);
for (j=info->ys; j<info->ys+info->ym; j++) {
    for (i=info->xs; i<info->xs+info->xm; i++) {
        f[j][i] = ... u[j][i] ... u[j-1][i] ... 
    }
}
```

Multigrid Structured Mesh Support:
Sample Jacobian Computation

```c
int Jacobian (DALocalInfo *info, double **u, Mat J, AppCtx *user)
MatStencil mrow, mcols[5];
double v[5];
...
for (j=info->ys; j<info->ys+info->ym; j++) {
    row.j = j;
    for (i=info->xs; i<info->xs+info->xm; i++) {
        v[0] = ...; col[0].j = j - 1; col[0].i = i;
        v[1] = ...; col[1].j = j; col[1].i = i-1;
        MatSetValuesStencil(jac, 1, &row, 5, col, v, INSERT_VALUES);
    }
}
```
Multigrid Structured Mesh Support:

Nonlinear Example

- 2-dim nonlinear problem with 4 degrees of freedom per mesh point
- Function() and Jacobian() are user-provided functions

DMMG dmmg;
DMMGCreate(comm,levels,rankuser,&dmmg);
DACreate2d(comm,DA_NONPERIODIC,DA_STENCIL_STAR,4,4,PETSC_DECIDE,PETSC_DECIDE,4,0,0,&da);
DMMGSetDM(dmmg,da);
DMMGSetSNESLocal(dmmg,Function,Jacobian,0,0);
DMMGSolve(dmmg);
solution = DMMGGetx(damg);

- All standard SNES, SLES, PC and MG options apply

Multigrid Structured Mesh Support:

Jacobian via Automatic Differentiation

- Collaboration with P. Hovland and B. Norris (see http://www.mcs.anl.gov/autodiff)
- Additional alternatives
  - Compute sparse Jacobian explicitly using AD
    DMMGSetSNESLocal(dmmg,Function(Jacobian),0,0)
    PETSc + ADIC automatically generate ad Function
  - Provide a “matrix-free” application of the Jacobian using AD
    DMMGSetSNESLocal(dmmg,Jacobian,0,0,ad Function)
    PETSc + ADIC automatically generate ad Function
- Similar situation for Fortran and ADIFOR

Using PETSc with Other Packages

- Linear algebra solvers
  - AMG
  - BlockSolve95
  - ILUTP
  - LUSOL
  - SPAI
  - SuperLU
- Mesh and discretization tools
  - Overture
  - SAMRAI
  - SUMA3d
- ODE solvers
  - PVODE
- Others
  - Matlab
  - PetMetis

Using PETSc with Other Packages:

Linear Solvers

- Interface Approach
  - Based on interfacing at the matrix level, where external linear solvers typically use
    a variant of compressed sparse row matrix storage
- Usage
  - Install PETSc; indicating presence of any optional external packages in the file
    petsc/bmake/PETSc_ARCH=base/site e.g.,
    - PETSC_HAVE_SPAI = -DPETSC_HAVE_SPAI
    - SPAI_INCLUDE = -I/home/username/software/spai_3.0/include
    - SPAI_LIB = /home/username/software/spai_3.0/lib/*PETSc_ARCH*/libspai.a
  - Set preconditioners via the usual approach
    - Procedure interface: PCSetType(pc,”spa1”)
    - Run time options: -pc_type spa1
  - Set preconditioner specific options via the usual approach, e.g.,
    - -pc_spa1_equality <eps>   -pc_spa1_verbose etc.

Using PETSc with Other Packages:

Linear Solvers (cont.)

- LUSOL
  - Sparse LU, part of MINOS
  - M. Saunders (Stanford Univer.)
  - http://www.sbsi-sol-optimize.com
  - PETSc interface by T. Munson (ANL), uses MatSeqAIJ
- SPAI
  - Sparse approximate inverse code by S. Barnhard (NASA Ames) and M. Grote (ETH Zurich)
  - http://www.sam.math.ethz.ch/~grote/spai
  - PETSc interface converts from any matrix format to SPAI matrix
- SuperLU
  - Parallel, sparse LU
  - J. Dongarra, J. Gilbert, (U.C. Berkeley) and X. Li (NREL)
  - http://www.netlib.org/superlu
  - PETSc interface uses MatSeqAIJ
  - Currently only sequential interface supported; parallel interface under development
Using PETSc with Other Packages: 

TAO – Optimization Software

- **TAO** - Toolkit for Advanced Optimization
  - Software for large-scale optimization problems
  - S. Benson, L. McInnes, and J. Moré
  - [http://www.mcs.anl.gov/tao](http://www.mcs.anl.gov/tao)
- **Initial TAO design uses PETSc for**
  - Low-level system infrastructure - managing portability
  - Parallel linear algebra tools (SLES)
    - [http://www.cs.umn.edu/~brennm/SLES](http://www.cs.umn.edu/~brennm/SLES)
  - Veltisto (library for PDE-constrained optimization by G. Biros, see [http://www.cs.nyu.edu/~biros/veltisto](http://www.cs.nyu.edu/~biros/veltisto)) – uses a similar interface approach
- **TAO is evolving toward**
  - CCA-compliant component-based design (see [http://www.cca-forum.org](http://www.cca-forum.org))
  - Support for ESI interfaces to various linear algebra libraries (see [http://z.ca.sandia.gov/esi](http://z.ca.sandia.gov/esi))

Using PETSc with Other Packages: 

PVODE – ODE Integrators

- **PVODE** – Parallel, robust, variable-order stiff and non-stiff ODE integrators
  - A. Hindmarsh et al. (LLNL)
  - [http://www.llnl.gov/CASC/PVODE](http://www.llnl.gov/CASC/PVODE)
  - L. Xu developed PVODE/PETSc interface
- **Interface Approach**
  - PVODE
    - ODE integrator – evolves field variables in time
    - `vector` – holds field variables
    - `preconditioner placeholder`
  - PETSc
    - ODE integrator placeholder
    - `vector`
    - sparse matrix and preconditioner
- **Usage**
  - `TSCreate(MPI_Comm,TS_NONLINEAR,&ts)`
  - `TSSetType(ts,TS_PVODE)`
  - `TSPVODESetType(ts,PVODE_ADAMS)`
  - other PVODE options
  - `TSSetFromOptions(ts)` – accepts PVODE options

Using PETSc with Other Packages: 

Mesh Management and Discretization

- **SUMAA3d** – Scalable Unstructured Mesh Algorithms and Applications
  - L. Freitag (ANL), M. Jones (VA Tech), P. Plassmann (Penn State)
  - L. Freitag and M. Jones developed SUMAA3d/PETSc interface
- **SAMRAI** – Structured adaptive mesh refinement
  - R. Hornung, S. Kohn (LLNL)
  - [http://www.llnl.gov/CASC/SAMRAI](http://www.llnl.gov/CASC/SAMRAI)
  - SAMRAI team developed SAMRAI/PETSc interface
- **Overture** – Structured composite meshes and discretizations
  - D. Brown, W. Henshaw, D. Quinlan (LLNL)
  - [http://www.llnl.gov/CASC/Overture](http://www.llnl.gov/CASC/Overture)
  - K. Buschelman and Overture team developed Overture/PETSc interfaces

Using PETSc with Other Packages: 

Matlab

- **Matlab**
  - [http://www.mathworks.com](http://www.mathworks.com)
- **Interface Approach**
  - PETSc socket interface to Matlab
  - Sends matrices and vectors/communicative Matlab sessions
  - PETSc interface to MatlabEngine
    - `MatlabEngine` – Matlab Engine that allows C/Fortran programmers to reuse Matlab functions in programs
    - `PetscMatlabEngine` – wraps PETSc vectors and matrices so that MatlabEngine can understand them
- **Usage**
  - `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: 

ParMETIS – Graph Partitioning

- **ParMETIS** – Parallel graph partitioning
  - G. Karypis (Univ. of Minnesota)
- **Interface Approach**
  - Use PETSc `MatPartitioning()` interface and MPIAIJ or MPIAdj matrix formats
- **Usage**
  - `MatPartitioningCreate(MPI_Comm,Mat Partitioning ctxt)`
  - `MatPartitioningSetAdjacency(ctxt,mat)`
  - `MatPartitioningSetVertexWeights(ctxt,weights)`
  - `MatPartitioningSetFromOptions(ctxt)`
  - `MatPartitioningApply(ctxt,IS *partitioning)`

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
  - preconditions: 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 and performance issues by writing to petsc-maint@mcs.anl.gov.

References

  - PETSc Users manual
  - Manual pages
  - Many hyperlinked examples
  - FAQ, Troubleshooting info, installation info, etc.
  - Research and publications that make use of PETSc
- MPI Information: [http://www.mpi-forum.org](http://www.mpi-forum.org)
  - *Using MPI* (2nd Edition), by Gropp, Lusk, and Skjellum
  - *Domain Decomposition*, by Smith, Bjorstad, and Gropp