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

This course is an excerpt of the 1/2-day PETSc tutorial at the Workshop on the ACTS Toolkit at NERSC, http://www-fp.mcs.anl.gov/petsc/docs/tutorials/nersc01/nersc01.htm, October 12, 2001.

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, HP, 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
  - Department of Energy: MICS Program, DOE2000, SciDAC
  - 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 Approach

From the perspective of an application programmer:

<table>
<thead>
<tr>
<th>Beginner</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic functionality, intended for use by most programmers</td>
<td>user-defined customization of algorithms and data structures</td>
</tr>
<tr>
<td>Emphasis of this tutorial</td>
<td></td>
</tr>
<tr>
<td>beginner</td>
<td>advanced</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Intermediate</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>selecting options, performance evaluation and tuning</td>
<td>advanced customizations, intended primarily for use by library developers</td>
</tr>
<tr>
<td>intermediate</td>
<td>developer</td>
</tr>
</tbody>
</table>

From the perspective of an application programmer:

- Beginner – basic functionality, intended for use by most programmers
- Intermediate – selecting options, performance evaluation and tuning
- Advanced – user-defined customization of algorithms and data structures
- Developer – advanced customizations, intended primarily for use by library developers
Structure of PETSc

PETSc PDE Application Codes

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

Computing and Communication Kernels
- MPI, MPI-IQ, BLAS, LAPACK

PETSc Numerical Components

### Nonlinear Solvers
- Newton-based Methods
- Line Search
- Trust Region
- Other

### Time Steppers
- Euler
- Backward Euler
- Pseudo Time Stepping
- Other

### Krylov Subspace Methods
- GMRES
- CG
- CGS
- Bi-CG-STAB
- TFQMR
- Richardson
- Chebychev
- Other

### Preconditioners
- Additive Schwarz
- Block Jacobi
- Jacobi
- ILU
- ICC
- LU (Sequential only)
- Others

### Matrices
- Compressed Sparse Row (AIJ)
- Blocked Compressed Sparse Row (BAIJ)
- Block Diagonal (BDIAG)
- Dense
- Matrix-free
- Other

### Distributed Arrays

### Vectors

### Index Sets
- Indices
- Block Indices
- Stride
- Other
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.

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;
}
```
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(...,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 \times x )</td>
</tr>
<tr>
<td>VecAYPX(Scalar *a, Vec x, Vec y)</td>
<td>( y = x + a \times y )</td>
</tr>
<tr>
<td>VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)</td>
<td>( w = a \times x + y )</td>
</tr>
<tr>
<td>VecScale(Scalar *a, Vec x)</td>
<td>( x = a \times 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 \times y_i )</td>
</tr>
<tr>
<td>VecMax(Vec x, int *idx, double *)</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 *)</td>
<td>( r =</td>
</tr>
</tbody>
</table>

**Data objects:** vectors

### 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 ...*

**Data objects:** vectors

**Beginner** - on-line exercise

---

October 12, 2001  
ACTS Toolkit Workshop
Matrices

• What are PETSc matrices?
  – Fundamental objects for storing linear operators (e.g., Jacobians)
• Create matrices via
  – MatCreate(…,Mat *)
    • 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 PDEs): MPIAIJ, SEQAIJ
      – symmetric block sparse AIJ: MPISBAIJ, SAEQSBAIJ
      – block diagonal: MPIBDIAG, SEQBDIAG
      – dense: MPIDENSE, SEQDENSE
      – matrix-free
      – etc.

Matrices and Polymorphism

• Single user interface, e.g.,
  – Matrix assembly
    • 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

```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);
```

simple 3-point stencil for 1D discretization
Parallel Matrix Distribution

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

\[
\begin{array}{c}
\text{proc 0} \\
\text{proc 1} \\
\text{proc 2} \\
\text{proc 3} \\
\text{proc 4}
\end{array}
\]

\{
proc 3: locally owned rows
\}

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

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, 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.

Solution components, using runtime option
-snes_vecmonitor

velocity: $u$

velocity: $v$

vorticity: $\zeta$

temperature: $T$
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

<table>
<thead>
<tr>
<th>Solver Classes</th>
<th>Usage Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear (SLES)</td>
<td>Context variables</td>
</tr>
<tr>
<td>Nonlinear (SNES)</td>
<td>Solver options</td>
</tr>
<tr>
<td>Timestepping (TS)</td>
<td>Callback routines</td>
</tr>
<tr>
<td></td>
<td>Customization</td>
</tr>
</tbody>
</table>
**Linear PDE Solution**

![Diagram showing the flow of control for PDE solution](image)

**Flow of Control for PDE Solution**

![Diagram showing the flow of control for PDE solution](image)
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 \( A, b \)

---

**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

**tutorial outline:**
- solvers: linear

---
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,PETSC_DECIDE,PETSC_DECIDE,
          n,n,&A);  /* assemble matrix */
VecCreate(MPI_COMM_WORLD,PETSC_DECIDE,n,&x);
VecDuplicate(x,&b); /* assemble RHS vector */
SLESCreate(MPI_COMM_WORLD,&sles);
SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN);
SLESSetOptions(sles);
SLESSolve(sles,b,x,&its);
SLESDestroy(sles);
```
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_ilu_levels <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,bcgts,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 ...

beginner

solvers: linear
preconditioners
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

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**

<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 iterations</td>
<td>PCSORSetIterations( )</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>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;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-sub_ksp_type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set convergence tolerances</td>
<td>KSPSetTolerances( )</td>
<td>-ksp_rtol &lt;rt&gt; -ksp_atol &lt;at&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_orthog</td>
</tr>
</tbody>
</table>

And many more options...

beginner  intermediate

solvers: linear: preconditioners

solvers: linear: Krylov methods
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 ...

tutorial outline: conclusion

Conclusion

- Summary
- 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

Using PETSc with Other Packages

- Linear algebra solvers
  - AMG
  - BlockSolve95
  - ILUTP
  - LUSOL
  - SPAI
  - SuperLU
- Optimization software
  - TAO
  - Veltisto
- Mesh and discretization tools
  - Overture
  - SAMRAI
  - SUMAA3d
- ODE solvers
  - PVODE
- Others
  - Matlab
  - ParMETIS
Using PETSc with Other Packages:
Linear Solvers

• **AMG**
  - Algebraic multigrid code by J. Ruge, K. Steuben, and R. Hempel (GMD)
  - [http://www.mgnet.org/mgnet-codes-gmd.html](http://www.mgnet.org/mgnet-codes-gmd.html)
  - PETSc interface by D. Lahaye (K.U.Leuven), uses MatSeqAIJ

• **BlockSolve95**
  - Parallel, sparse ILU(0) for symmetric nonzero structure and ICC(0)
  - M. Jones (Virginia Tech.) and P. Plassmann (Penn State Univ.)
  - [http://www.mcs.anl.gov/BlockSolve95](http://www.mcs.anl.gov/BlockSolve95)
  - PETSc interface uses MatMPIRowbs

• **ILUTP**
  - Drop tolerance ILU by Y. Saad (Univ. of Minnesota), in SPARSKIT
  - PETSc interface uses MatSeqAIJ

---

Using PETSc with Other Packages:
Linear Solvers (cont.)

• **LUSOL**
  - Sparse LU, part of MINOS
  - M. Saunders (Stanford Univ)
  - [http://www.sbsi-sol-optimize.com](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)
  - PETSc interface converts from any matrix format to SPAI matrix

• **SuperLU**
  - Parallel, sparse LU
  - J. Demmel, J. Gilbert, (U.C. Berkeley) and X. Li (NERSC)
  - [http://www.nersc.gov/~xiaoye/SuperLU](http://www.nersc.gov/~xiaoye/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)
  - Velletisto (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)
  - ….. regular TS functions
  - 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 to interactive Matlab session
  - PETSc interface to MatlabEngine
    - MatlabEngine = Matlab library that allows C/Fortran programmers to use Matlab functions in programs
    - PetscMatlabEngine = unwraps 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)
  – http://www.cs.umn.edu/~karypis/metis/parmetis
• Interface Approach
  – Use PETSc MatPartitioning() interface and MPIAIJ or MPIAdj matrix formats
• Usage
  – MatPartitioningCreate(MPI_Comm, MatPartitioning ctx)
  – MatPartitioningSetAdjacency(ctx, matrix)
  – Optional – MatPartitioningSetVertexWeights(ctx, weights)
  – MatPartitioningSetFromOptions(ctx)
  – MatPartitioningApply(ctx, 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
  – 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 and performance issues by writing to petsc-maint@mcs.anl.gov.

References

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