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 2.2.1 of PETSc
(updated, Aug. 2004 by Rolf Rabenseifner)

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:

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

Emphasis of this tutorial

1

beginner

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

3

advanced

- Intermediate
  - selecting options, performance evaluation and tuning

2

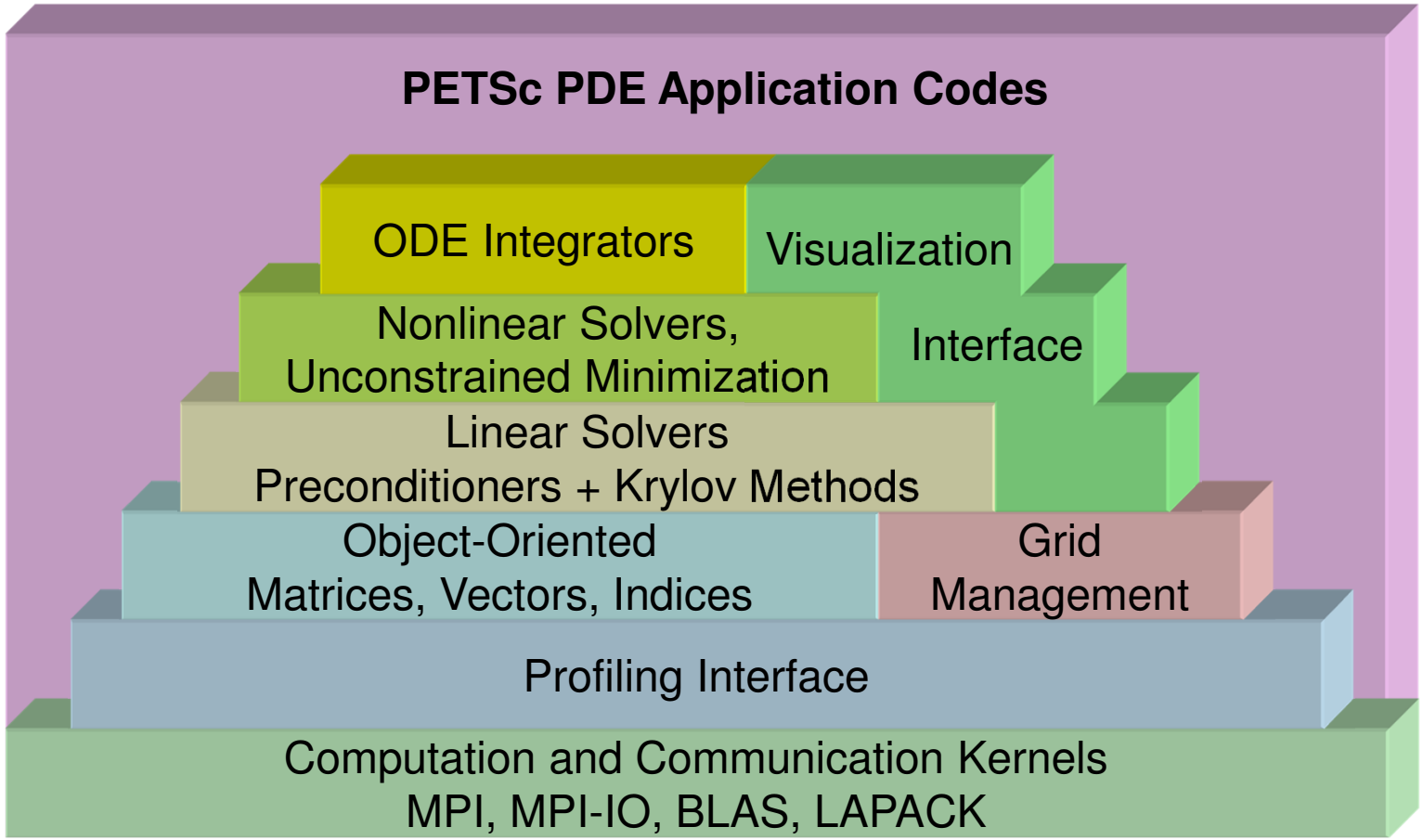
intermediate

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

4

developer

Structure of PETSc



PETSc Numerical Components

Nonlinear Solvers		
Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers			
Euler	Backward Euler	Pseudo Time Stepping	Other

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

Preconditioners						
Additive Schwartz	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

skipped

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

```
#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

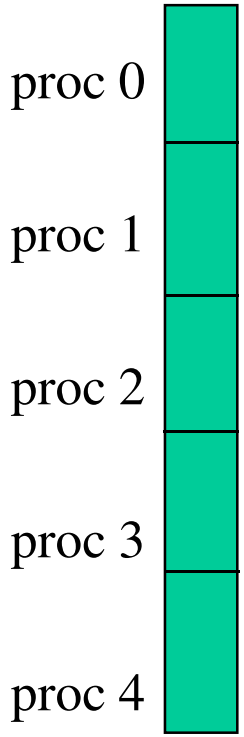
- Vectors (Vec)
  - focus: field data arising in nonlinear PDEs
- Matrices (Mat)
  - focus: linear operators arising in nonlinear PDEs (i.e., Jacobians)

beginner	• Object creation
beginner	• Object assembly
intermediate	• Setting options
intermediate	• Viewing
advanced	• User-defined customizations

tutorial outline:
data objects

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



beginner

data objects: vectors

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)

beginner

data objects: vectors

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

data objects: vectors and matrices

beginner

Selected Vector Operations

Function Name	Operation
VecAXPY(Scalar *a, Vec x, Vec y)	$y = y + a*x$
VecAYPX(Scalar *a, Vec x, Vec y)	$y = x + a*y$
VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)	$w = a*x + y$
VecScale(Scalar *a, Vec x)	$x = a*x$
VecCopy(Vec x, Vec y)	$y = x$
VecPointwiseMult(Vec x, Vec y, Vec w)	$w\_i = x\_i *y\_i$
VecMax(Vec x, int *idx, double *r)	$r = max\ x\_i$
VecShift(Scalar *s, Vec x)	$x\_i = s+x\_i$
VecAbs(Vec x)	$x\_i = \|x\_i\|$
VecNorm(Vec x, NormType type , double *r)	$r = \|\|x\|\|$

beginner

data objects:
vectors

skipped

Simple Example Programs

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

E ex2.c	- synchronized printing	1

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

E ex1.c, ex1f.F, ex1f90.F	- basic vector routines	1
E ex3.c, ex3f.F	- parallel vector layout	

And many more examples ...

E - on-line exercise

1 beginner

data objects: vectors

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.

beginner

data objects: matrices

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.

beginner

data objects: matrices

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)

beginner

data objects:
matrices

skipped

Matrix Assembly Example

simple 3-point stencil for 1D discretization

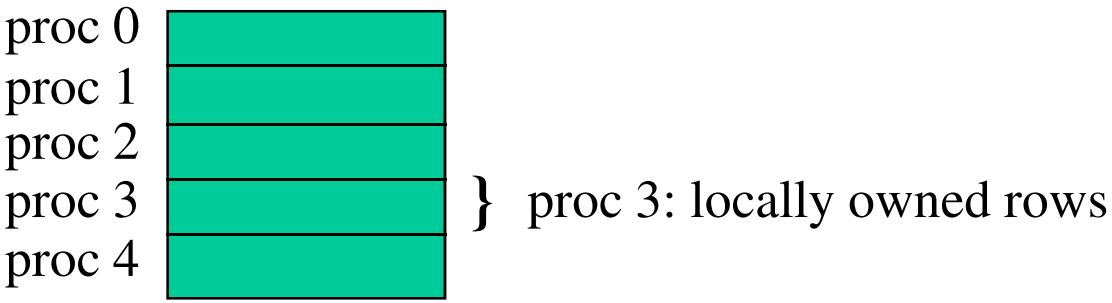
```
Mat     A;
int     column[3], i, start, end;
double value[3];

/* 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);
```

beginner

data objects: matrices

Parallel Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global 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

data objects: matrices

beginner

PETSc Viewers

Viewers

beginner
- Printing information about solver and data objects

beginner
- Visualization of field and matrix data

intermediate
- Binary output of vector and matrix data

tutorial outline: viewers

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

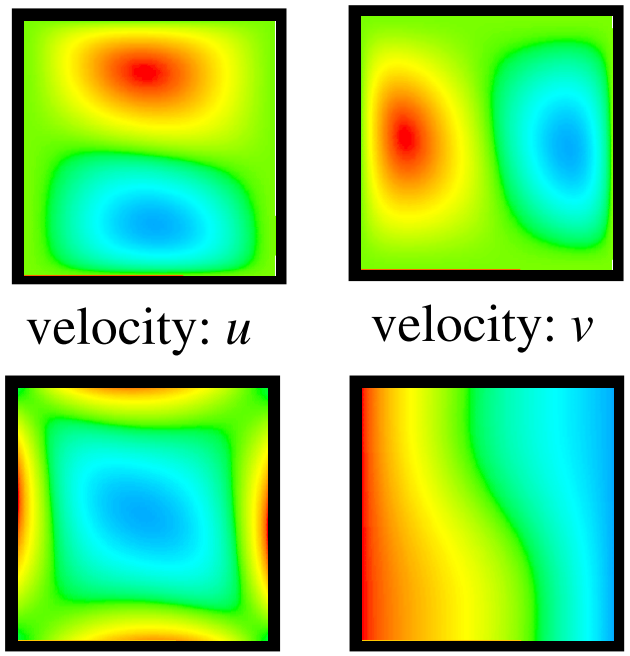
beginner

viewers

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

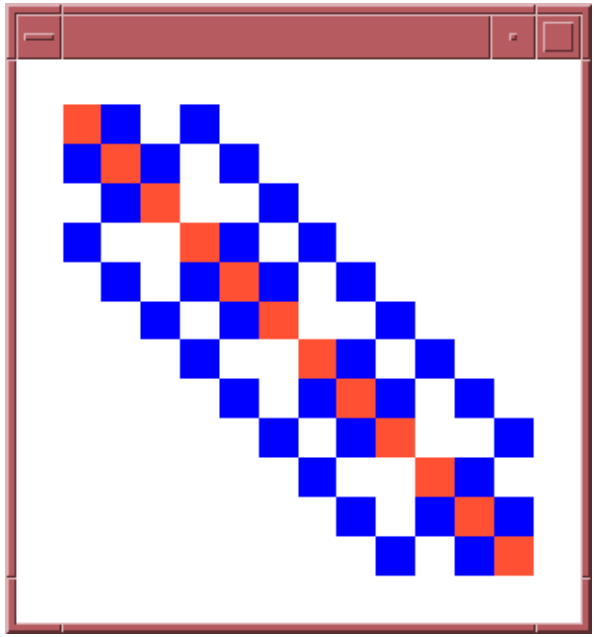
vorticity: ζ temperature: T

beginner

viewers

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.



beginner

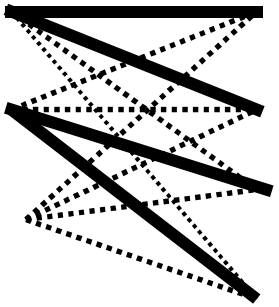
viewers

Solvers: Usage Concepts

Solvers

Solver Classes

- Linear (KSP)
- Nonlinear (SNES)
- Timestepping (TS)



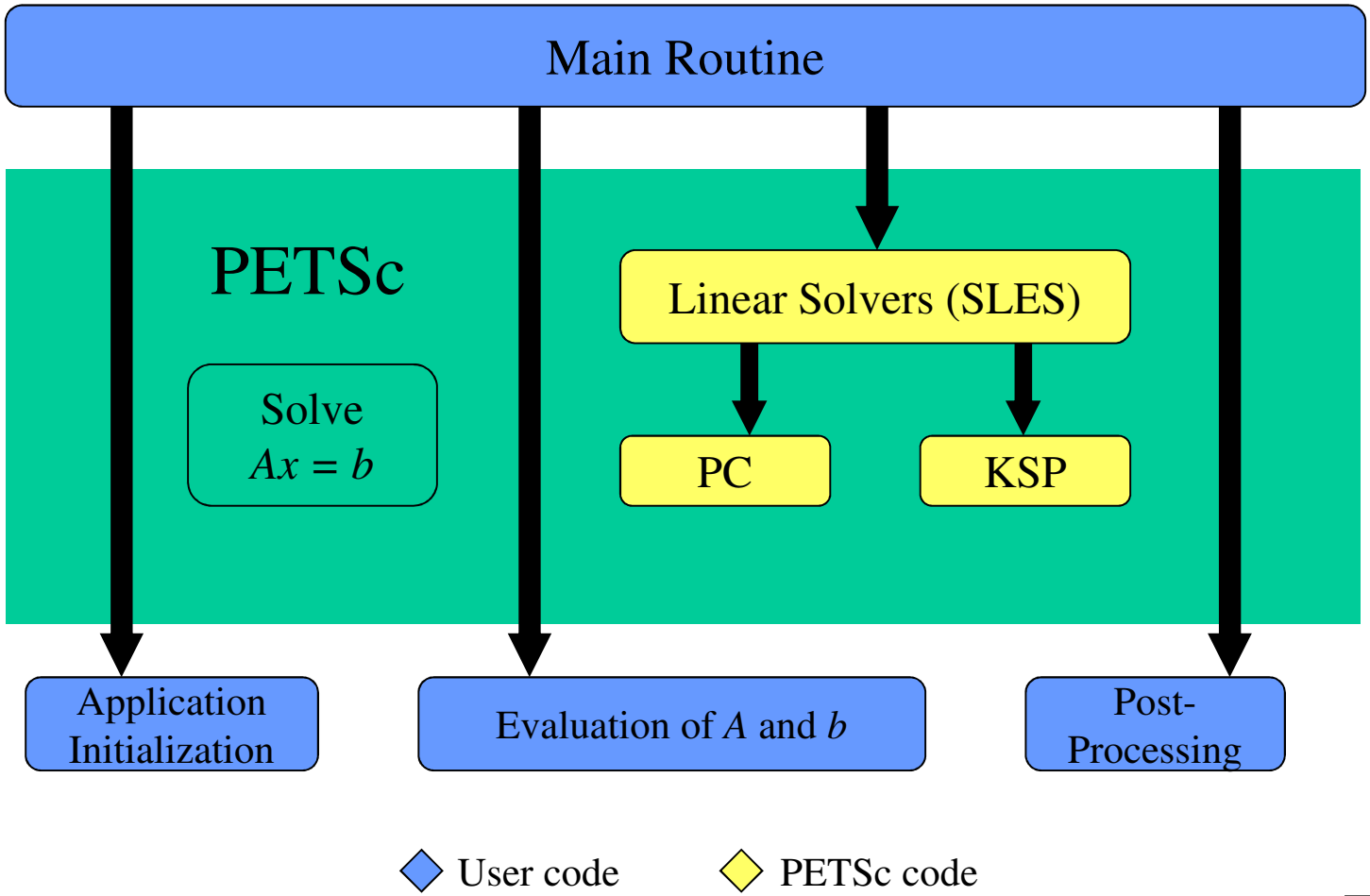
Usage Concepts

- Context variables
- Solver options
- Callback routines
- Customization

important concepts

tutorial outline: solvers

Linear PDE Solution

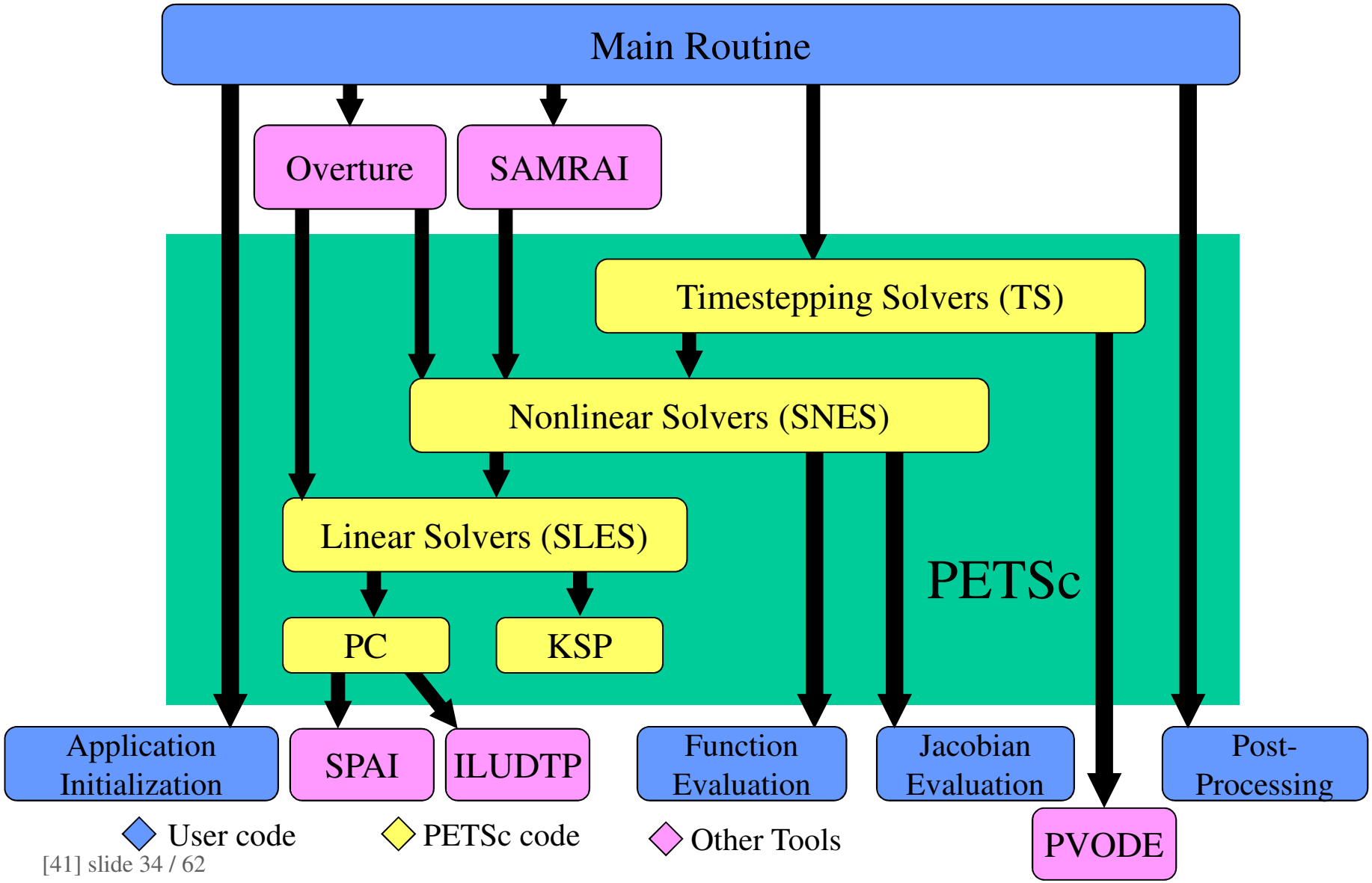


beginner

solvers: linear

skipped

Flow of Control for PDE Solution



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

beginner

solvers: linear

Linear Solvers (KSP)

KSP: Krylov Sub-Space Methods

beginner	• Application code interface
beginner	• Choosing the solver
beginner	• Setting algorithmic options
beginner	• Viewing the solver
intermediate	• Determining and monitoring convergence
intermediate	• Providing a different preconditioner matrix
advanced	• Matrix-free solvers
advanced	• 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)

beginner

solvers: linear

Creating the KSP Context

- C/C++ version

```
ierr = KSPCreate(MPI_COMM_WORLD,&ksp);
```

- Fortran version

```
call KSPCreate(MPI_COMM_WORLD,ksp,ierr)
```

- Provides an **identical** user interface for all linear solvers
  - uniprocessor and parallel
  - real and complex numbers

PETSc 2.1.6 and former:
***S**calable **L**inear **E**quations **S**olvers*
SLES context and SLES...() routines were used instead of KSP

beginner

solvers: linear

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.

beginner

solvers: linear

Basic Linear Solver Code (C/C++)

```
KSP    ksp;          /* Krylov sub-space (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 */

KSPCreate(MPI_COMM_WORLD,&ksp);
KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
KSPSetFromOptions(ksp);
/* KSPSetRhs(ksp,b);   KSPSetSolution(ksp,x); KSPSolve(ksp); PETSC 2.2.0 */
KSPSolve(ksp, b, x);  /* PETSC 2.2.1 */
KSPDestroy(ksp);
```

beginner

solvers: linear

Options

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

beginner

solvers: linear

skipped

Setting Solver Options within Code

- On ksp:
  - KSPSetType(KSP ksp,KSPType type)
  - KSPSetTolerances(KSP ksp,PetscReal rtol, PetscReal atol,PetscReal dtol, int maxits)
  - etc....
- KSPGetPC(KSP ksp,PC *pc)
  - PCSetType(PC pc,PCType)
  - PCASMSetOverlap(PC pc,int overlap)
  - etc....

beginner

solvers: linear

Recursion: Specifying Solvers for Schwarz Preconditioner Blocks

- Specify KSP&PC 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>

beginner

solvers: linear: preconditioners

Setting Solver Options at Runtime

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

1

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

2

1	2
beginner	intermediate

solvers:
linear

skipped

Linear Solvers: Monitoring Convergence

- -ksp_monitor - Prints preconditioned residual norm
- -ksp_xmonitor - Plots preconditioned residual norm

1

- -ksp_truemonitor - Prints true residual norm || b-Ax ||
- -ksp_xtruemonitor - Plots true residual norm || b-Ax ||

2

- User-defined monitors, using callbacks

3

1	2	3
beginner	intermediate	advanced

solvers: linear

Review

KSP: Review of Basic Usage

- KSPCreate( ) - Create KSP context
- KSPSetOperators( ) - Set linear operators
- KSPSetFromOptions( ) - Set runtime solver options for [KSP,PC]
- KSPSetRhs() - Set right hand side context /* 2.2.0 */
- KSPSetSolution() - Set solution vector context /* 2.2.0 */
- KSPSolve( ) - Run linear solver
- KSPView( ) - View solver options actually used at runtime (alternative: -ksp_view)
- KSPDestroy( ) - Destroy solver

beginner

solvers: linear

PC: Review of Selected Preconditioner Options

Functionality	Procedural Interface	Runtime Option	
Set preconditioner type	PCSetType( )	-pc_type [lu,ilu,jacobi, sor,asm,…]	1
Set level of fill for ILU	PCILUSetLevels( )	-pc_ilu_levels <levels>	2
Set SOR iterations	PCSORSetIterations( )	-pc_sor_its <its>	
Set SOR parameter	PCSORSetOmega( )	-pc_sor_omega <omega>	
Set additive Schwarz variant	PCASMSetType( )	-pc_asm_type [basic, restrict,interpolate,none]	
Set subdomain solver options	PCGetSubSLES( )	-sub_pc_type <pctype> -sub_ksp_type <ksptype> -sub_ksp_rtol <rtol>	

And many more options…

1	2
beginner	intermediate

solvers: linear: preconditioners

KSP: Review of Selected Krylov Method Options

Functionality	Procedural Interface	Runtime Option
Set Krylov method	KSPSetType( )	-ksp_type [cg,gmres,bcgs, tfqmr,cgs,…] 1
Set monitoring routine	KSPSetMonitor( )	-ksp_monitor, –ksp_xmonitor, -ksp_truemonitor, -ksp_xtruemonitor
Set convergence tolerances	KSPSetTolerances( )	-ksp_rtol <rt> -ksp_atol <at> -ksp_max_its <its> 2
Set GMRES restart parameter	KSPGMRESSetRestart( )	-ksp_gmres_restart <restart>
Set orthogonalization routine for GMRES	KSPGMRESSet Orthogonalization( )	-ksp_unmodifiedgramschmidt -ksp_irorthog

And many more options...

1	2
beginner	intermediate

solvers: linear: Krylov methods

skipped

KSP: Example Programs

Location: petsc/src/ksp/examples/tutorials/

• ex1.c, ex1f.F E ex23.c • ex11.c	- basic uniprocessor codes - basic parallel code - using complex numbers	1
• ex4.c • ex9.c E ex22.c	- using different linear system and preconditioner matrices - repeatedly solving different linear systems - 3D Laplacian using multigrid	2
• ex15.c	- setting a user-defined preconditioner	3

And many more examples ...

1	2	3
beginner	intermediate	advanced

E - on-line exercise

solvers: linear

Conclusion

Conclusion

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

tutorial outline:
conclusion

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

skipped

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
  - 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
  - PETSc interface uses MatMPIRowbs
- ILUTP
  - Drop tolerance ILU by Y. Saad (Univ. of Minnesota), in SPARSKIT
  - http://www.cs.umn.edu/~saad/
  - PETSc interface uses MatSeqAIJ

skipped

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

- LUSOL
  - Sparse LU, part of MINOS
  - M. Saunders (Stanford Univ)
  - 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. Demmel, J. Gilbert, (U.C. Berkeley) and X. Li (NERSC)
  - http://www.nersc.gov/~xiaoye/SuperLU
  - PETSc interface uses MatSeqAIJ
  - Currently only sequential interface supported; parallel interface under development

skipped

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
- Initial TAO design uses PETSc for
  - Low-level system infrastructure - managing portability
  - Parallel linear algebra tools (KSP, formerly(up to petsc-2.1.6) SLES)
    - Veltisto (library for PDE-constrained optimization by G. Biros, see 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)
  - Support for ESI interfaces to various linear algebra libraries (see http://z.ca.sandia.gov/esi)

skipped

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
  - 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

skipped

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)
  - http://www.mcs.anl.gov/sumaa3d
  - 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
  - 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
  - K. Buschelman and Overture team developed Overture/PETSc interfaces

Using PETSc with Other Packages: Matlab

- Matlab
  - 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/ksp/ksp/impls/cg
  - preconditioners: petsc/src/ksp/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/ → Documentation
  - 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 of PETSc
- MPI Information: http://www.mpi-forum.org
- ***Using MPI*** (2nd Edition), by Gropp, Lusk, and Skjellum
- ***Domain Decomposition***, by Smith, Bjorstad, and Gropp