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, 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

The PETSc Team

Satish Balay

Matt Knepley

Kris Buschelman

Lois Curfman McInnes

Bill Gropp

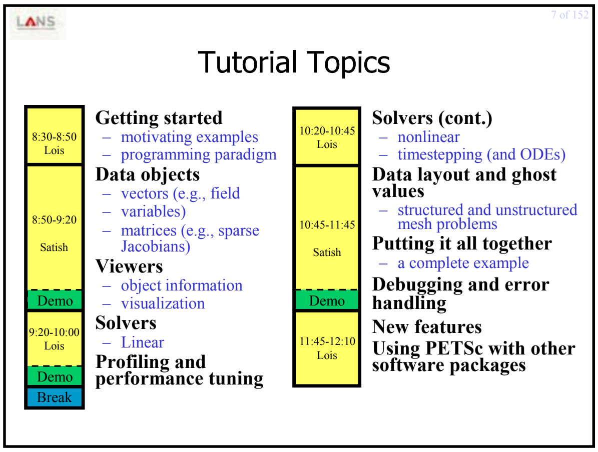
Barry Smith

Dinesh Kaushik

Hong Zhang

PETSc Concepts

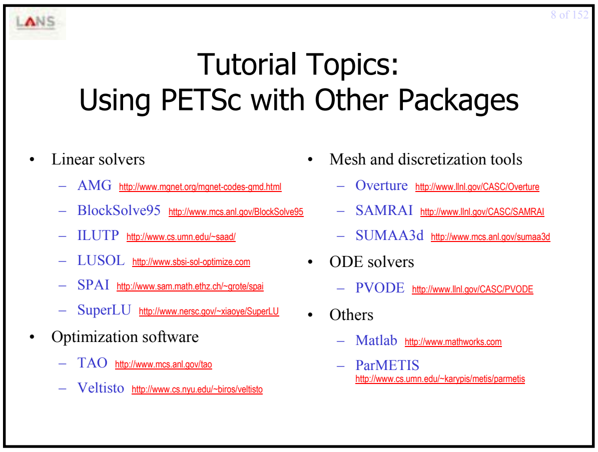
- How to specify the mathematics of the problem
  - Data objects
    - vectors, matrices
- How to solve the problem
  - Solvers
    - linear, nonlinear, and time stepping (ODE) solvers
- Parallel computing complications
  - Parallel data layout
    - structured and unstructured meshes



LANS

Tutorial Topics

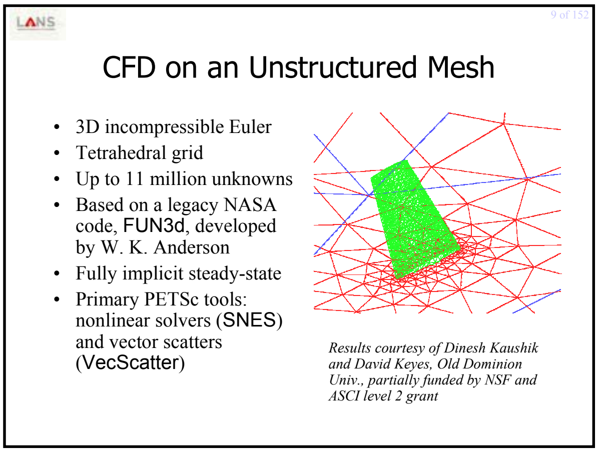
8:30-8:50 Lois	**Getting started** – motivating examples – programming paradigm
8:50-9:20 Satish	**Data objects** – vectors (e.g., field variables) – matrices (e.g., sparse Jacobians) **Viewers** – object information – visualization
Demo	
9:20-10:00 Lois	**Solvers** – Linear **Profiling and performance tuning**
Demo	
Break	
10:20-10:45 Lois	**Solvers (cont.)** – nonlinear – timestepping (and ODEs)
10:45-11:45 Satish	**Data layout and ghost values** – structured and unstructured mesh problems **Putting it all together** – a complete example
Demo	**Debugging and error handling**
11:45-12:10 Lois	**New features** **Using PETSc with other software packages**



LANS

Tutorial Topics: Using PETSc with Other Packages

- Linear solvers
  - AMG http://www.mgnet.org/mgnet-codes-gmd.html
  - BlockSolve95 http://www.mcs.anl.gov/BlockSolve95
  - ILUTP http://www.cs.umn.edu/~saad/
  - LUSOL http://www.sbsi-sol-optimize.com
  - SPAI http://www.sam.math.ethz.ch/~grote/spai
  - SuperLU http://www.nersc.gov/~xiaoye/SuperLU
- Optimization software
  - TAO http://www.mcs.anl.gov/tao
  - Veltisto http://www.cs.nyu.edu/~biros/veltisto
- Mesh and discretization tools
  - Overture http://www.llnl.gov/CASC/Overture
  - SAMRAI http://www.llnl.gov/CASC/SAMRAI
  - SUMAA3d http://www.mcs.anl.gov/sumaa3d
- ODE solvers
  - PVODE http://www.llnl.gov/CASC/PVODE
- Others
  - Matlab http://www.mathworks.com
  - ParMETIS http://www.cs.umn.edu/~karypis/metis/parmetis

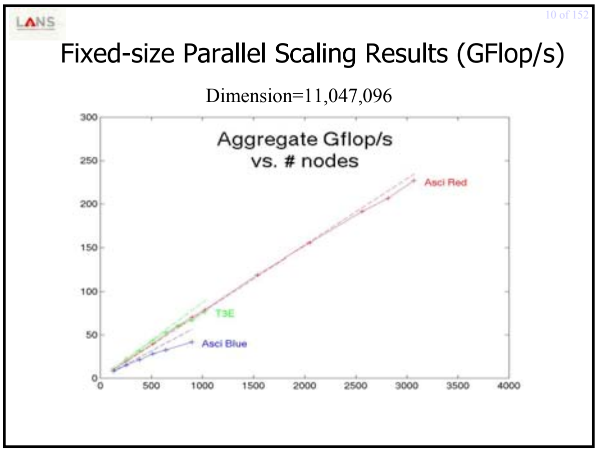


LANS

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)

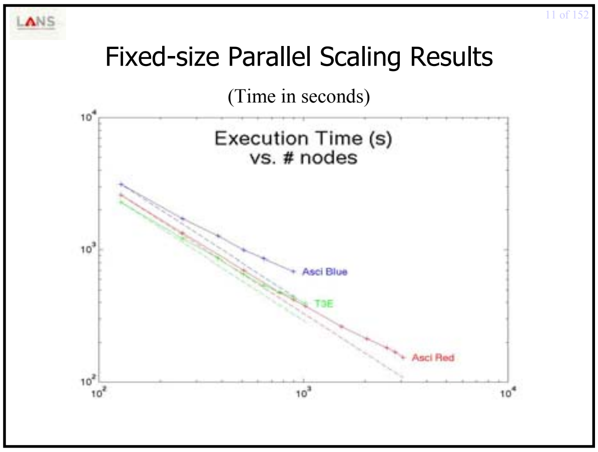
Results courtesy of Dinesh Kaushik and David Keyes, Old Dominion Univ., partially funded by NSF and ASCI level 2 grant



LANS

Fixed-size Parallel Scaling Results (GFlop/s)

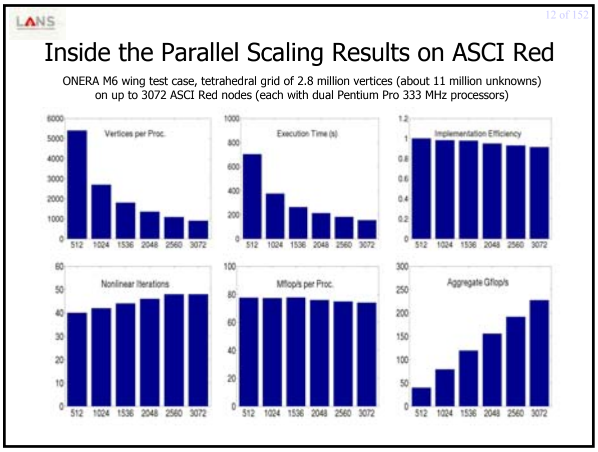
Dimension=11,047,096



LANS

Fixed-size Parallel Scaling Results

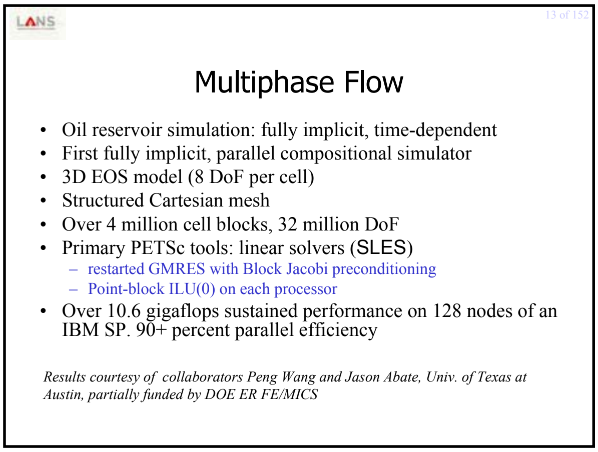
(Time in seconds)



LANS

Inside the Parallel Scaling Results on ASCI Red

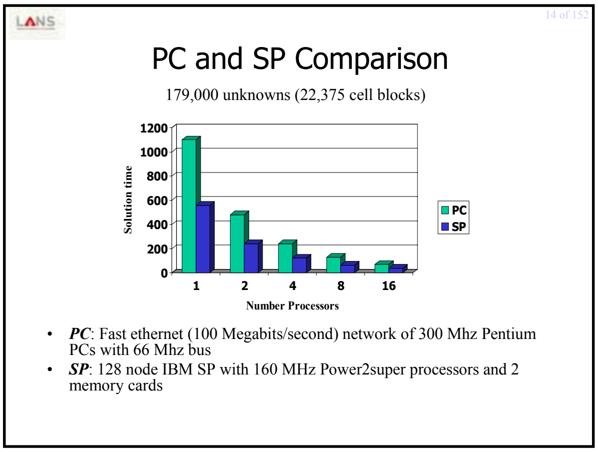
ONERA M6 wing test case, tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on up to 3072 ASCI Red nodes (each with dual Pentium Pro 333 MHz processors)



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

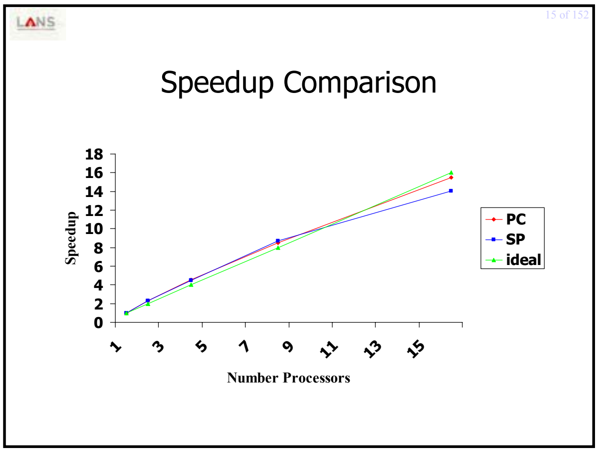
Results courtesy of collaborators Peng Wang and Jason Abate, Univ. of Texas at Austin, partially funded by DOE ER FE/MICS



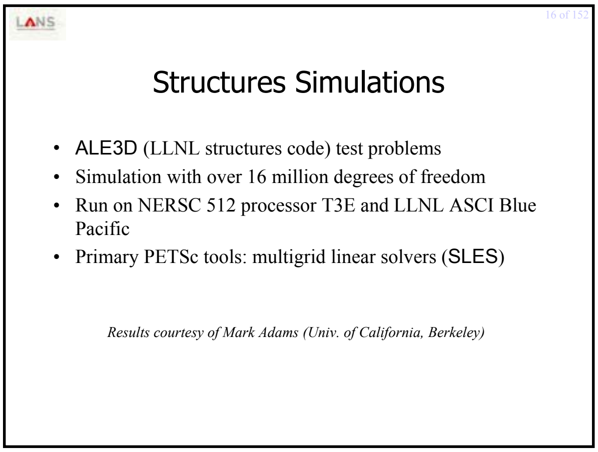
PC and SP Comparison

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



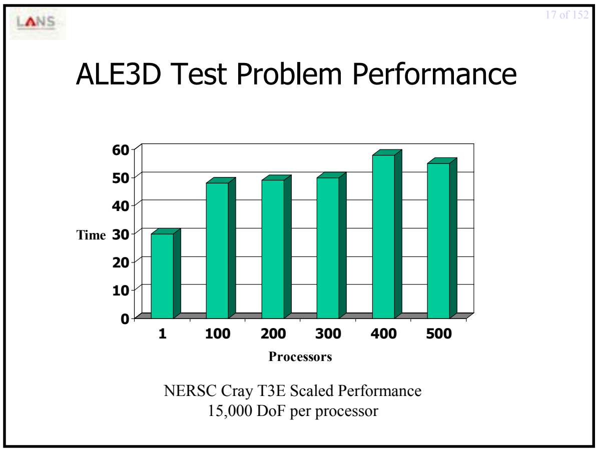
Speedup Comparison



Structures Simulations

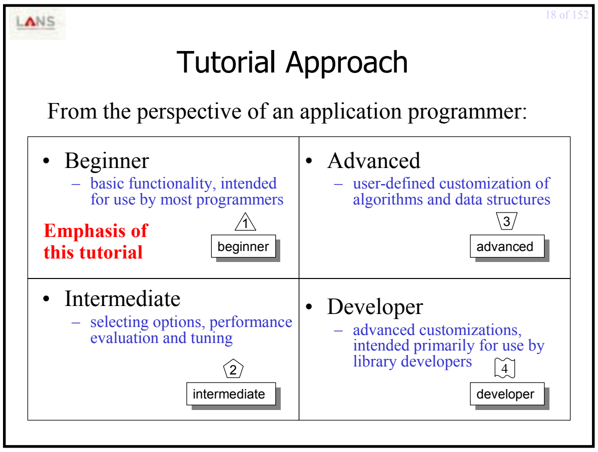
- ALE3D (LLNL structures code) test problems
- Simulation with over 16 million degrees of freedom
- Run on NERSC 512 processor T3E and LLNL ASCI Blue Pacific
- Primary PETSc tools: multigrid linear solvers (SLES)

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

1
beginner

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

3
advanced

- Intermediate
  - selecting options, performance evaluation and tuning

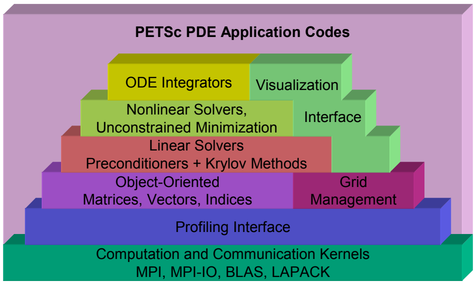
2
intermediate

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

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

Structure of PETSc



PETSc PDE Application Codes

ODE Integrators

Visualization

Nonlinear Solvers, Unconstrained Minimization

Interface

Linear Solvers Preconditioners + Krylov Methods

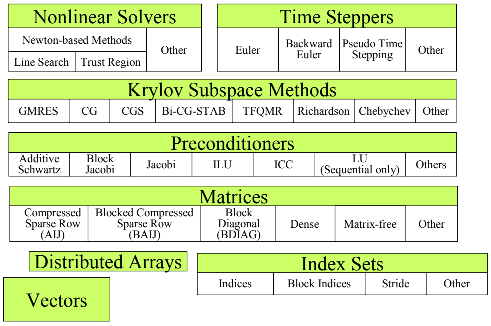
Object-Oriented Matrices, Vectors, Indices

Grid Management

Profiling Interface

Computation and Communication Kernels MPI, MPI-IO, BLAS, LAPACK

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

Index Sets			
Indices	Block Indices	Stride	Other

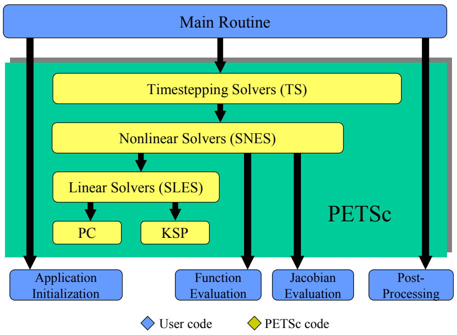
Vectors

What is not in PETSc?

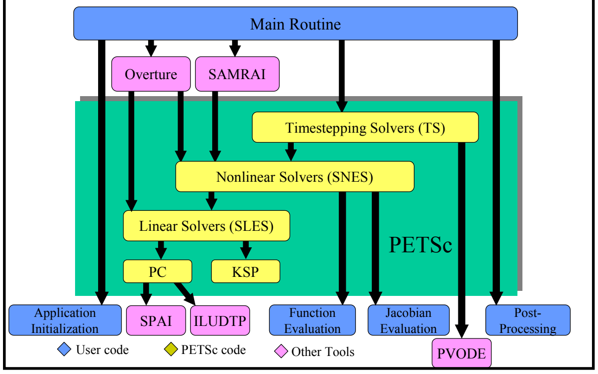
- Discretizations
- Unstructured mesh generation and refinement tools
- Load balancing tools
- Sophisticated visualization capabilities

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

Flow of Control for PDE Solution



Flow of Control for PDE Solution



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

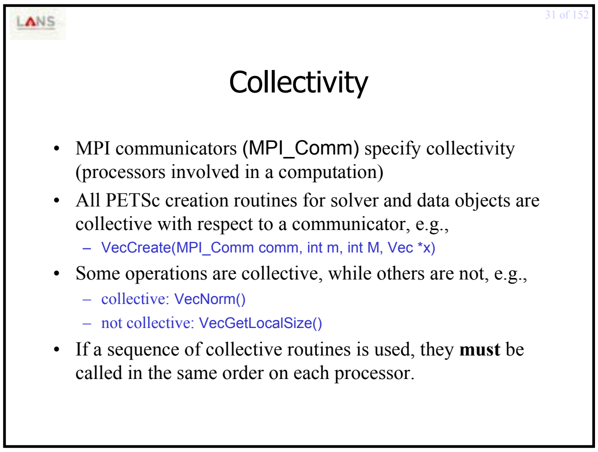
- Encapsulate the latest numerical algorithms in a consistent, application-friendly manner
- Use mathematical and algorithmic objects, not low-level programming language objects
- Application code focuses on mathematics of the global problem, not parallel programming details

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

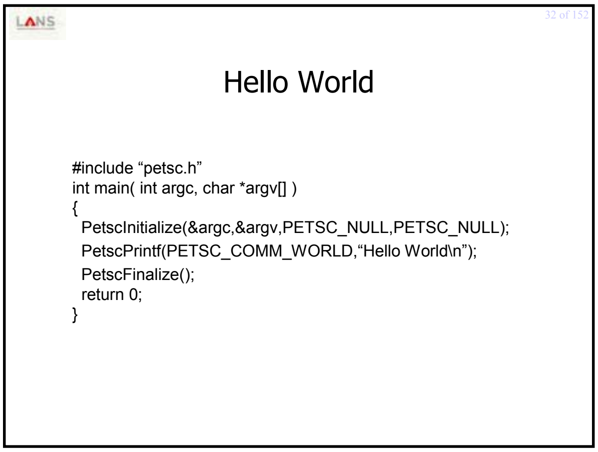
The PETSc Programming Model

- **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



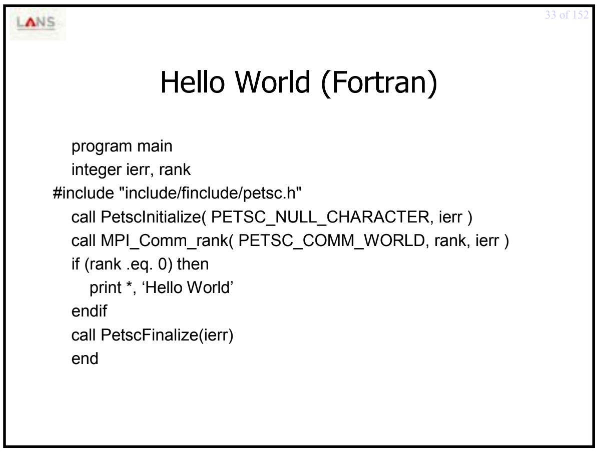
Collectivity

- 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.



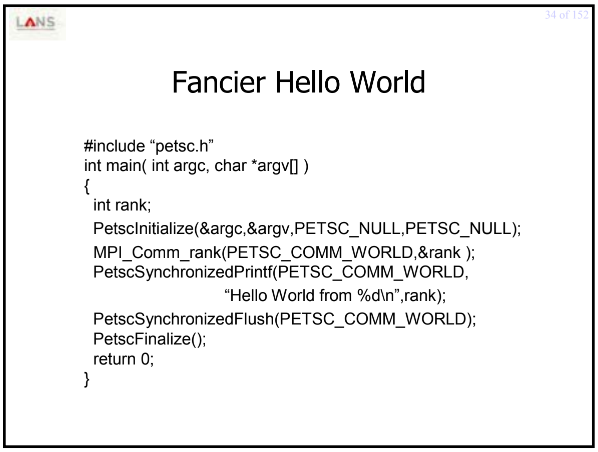
Hello World

```
#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;
}
```



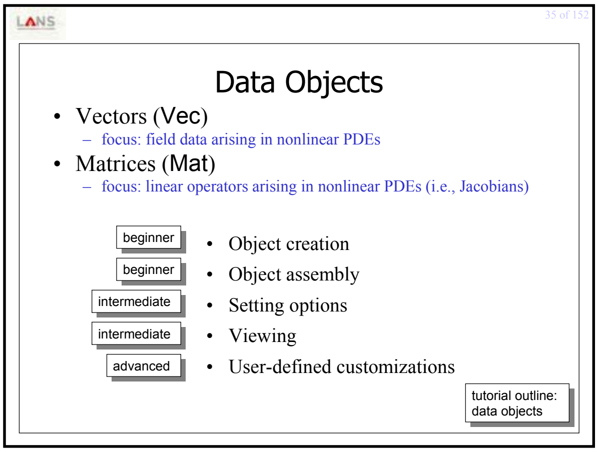
Hello World (Fortran)

```
    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

```
#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)

beginner
- Object creation

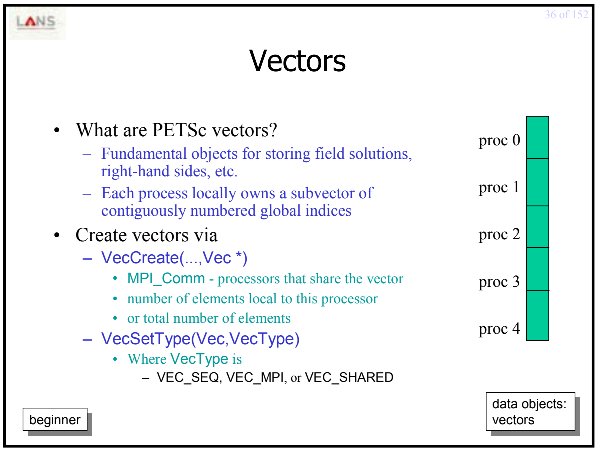
beginner
- Object assembly

intermediate
- Setting options

intermediate
- Viewing

advanced
- User-defined customizations

tutorial outline:
data objects



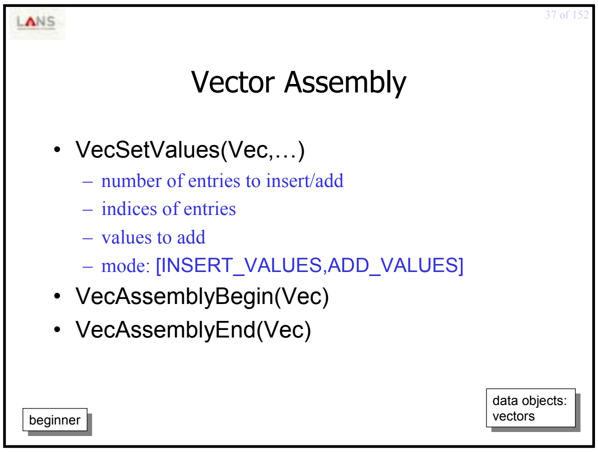
Vectors

- 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

proc 0
proc 1
proc 2
proc 3
proc 4

beginner

data objects:
vectors

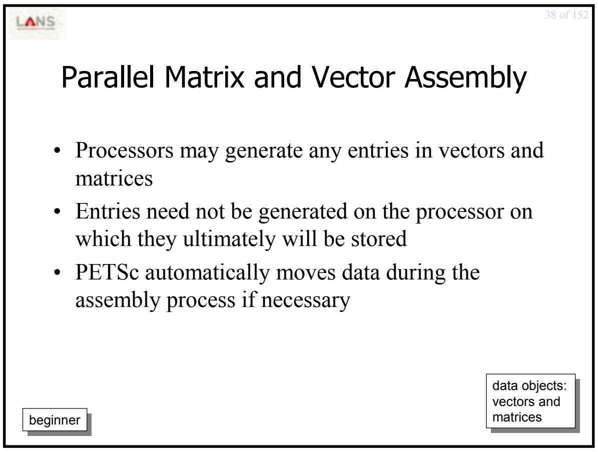


Vector Assembly

- 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

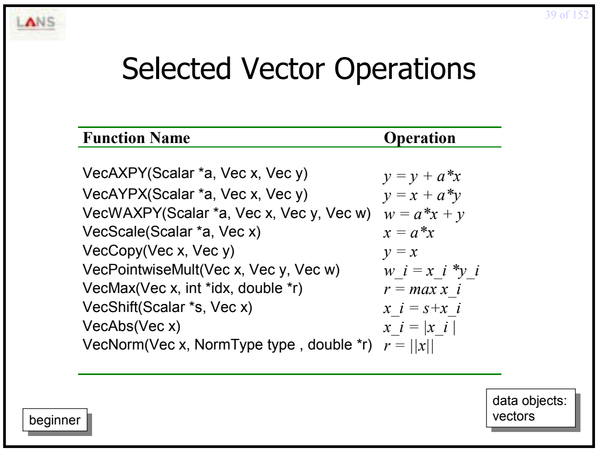


Parallel Matrix and Vector Assembly

- 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

beginner

data objects: vectors and matrices

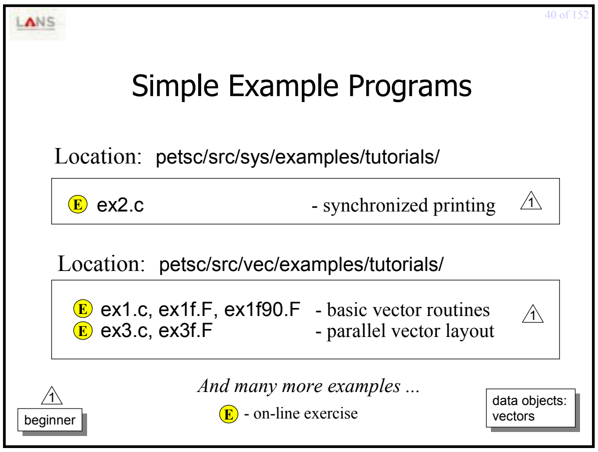


Selected Vector Operations

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



Simple Example Programs

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

Ⓔ ex2.c - synchronized printing ⚠1

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

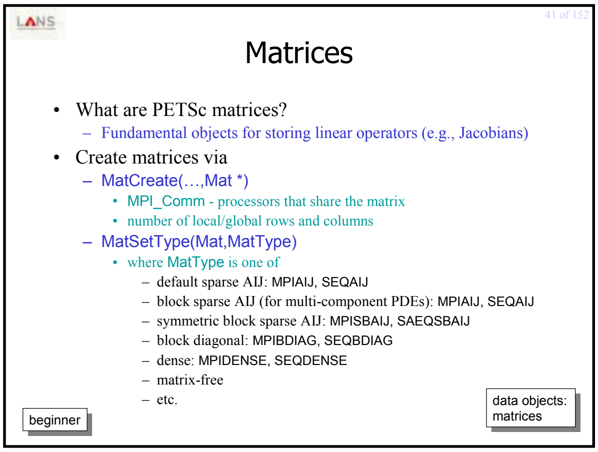
Ⓔ ex1.c, ex1f.F, ex1f90.F - basic vector routines ⚠1
Ⓔ ex3.c, ex3f.F - parallel vector layout

And many more examples ...

Ⓔ - on-line exercise

⚠1 beginner

data objects: vectors

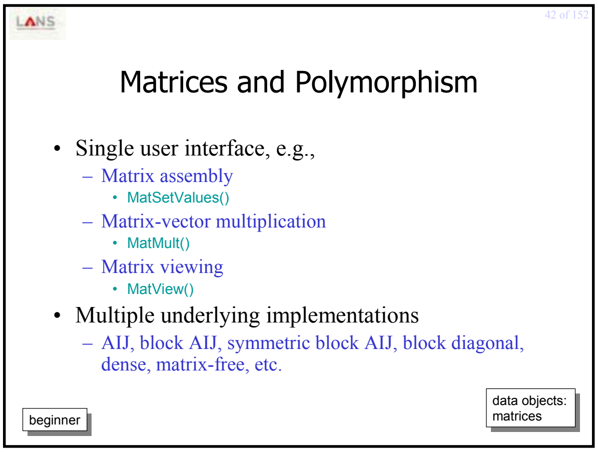


Matrices

- 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

Matrix Assembly Example

simple 3-point stencil for 1D discretization

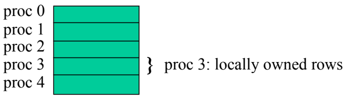
```
Mat     A;
int      column[3], i, start, end;
double value[3];

/* 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)

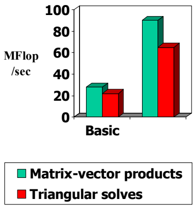
- rstart: first locally owned row of global matrix
- rend -1: last locally owned row of global matrix

beginner

data objects: matrices

Blocking: Performance Benefits

More issues discussed in full tutorials available via PETSc web site.



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

beginner

data objects: matrices

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

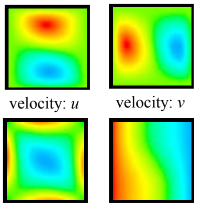
beginner

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



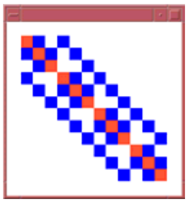
velocity: *u* velocity: *v*

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.

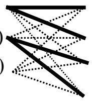


beginner viewers

Solvers: Usage Concepts

Solver Classes

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



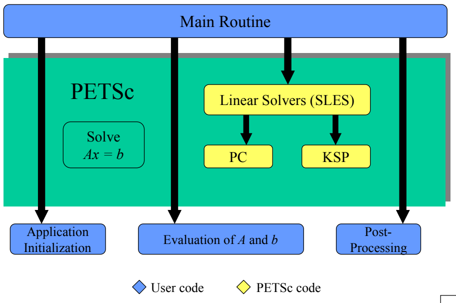
Usage Concepts

- Context variables
- Solver options
- Callback routines
- Customization

important concepts

tutorial outline: solvers

Linear PDE Solution



Main Routine

PETSc

Solve $Ax = b$

Linear Solvers (SLES)

PC

KSP

Application Initialization

Evaluation of A and b

Post-Processing

◆ User code ◇ PETSc code

beginner solvers: linear

Linear Solvers

Goal: Support the solution of linear systems,

Ax=b,

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

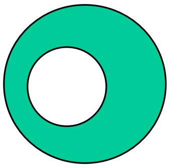
User provides:

- Code to evaluate *A, b*

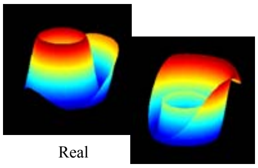
beginner solvers: linear

Sample Linear Application:

Exterior Helmholtz Problem



Solution Components



Real

Imaginary

$$-\nabla^2 u - k^2 u = 0$$

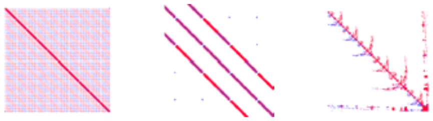
$$\lim_{r\to\infty} r^{1/2}\left(\frac{\partial u}{\partial r} + iku\right) = 0$$

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

beginner solvers: linear

Helmholtz: The Linear System

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



Natural ordering Close-up Nested dissection ordering

beginner

solvers: linear

Linear Solvers (SLES)

SLES: Scalable Linear Equations Solvers

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 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

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

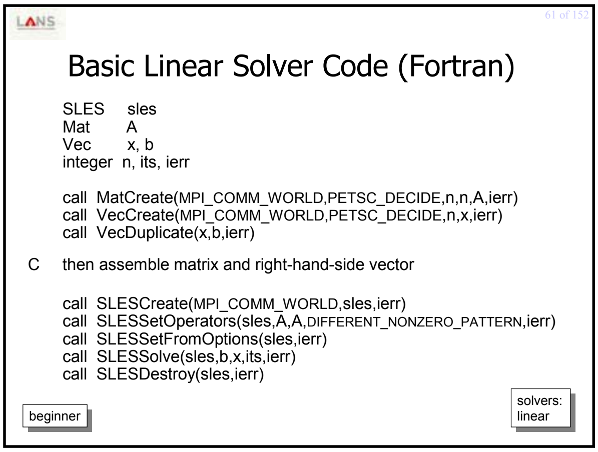
solvers: linear

Basic Linear Solver Code (C/C++)

```
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);
SLESSetFromOptions(sles);
SLESSolve(sles,b,x,&its);
SLESDestroy(sles);
```

beginner

solvers: linear



LANS

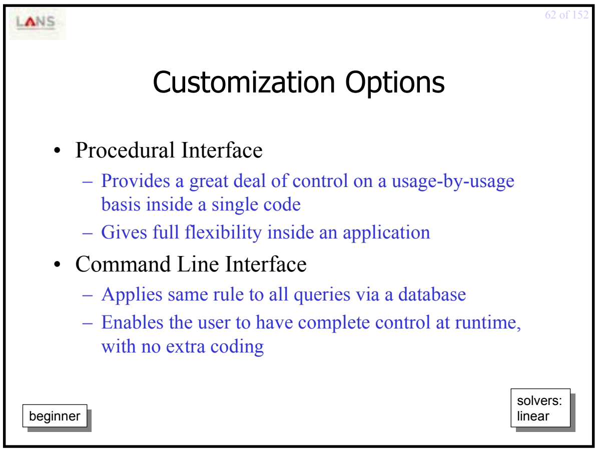
61 of 152

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  SLESCreate(MPI_COMM_WORLD,sles,ierr)
      call  SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN,ierr)
      call  SLESSetFromOptions(sles,ierr)
      call  SLESSolve(sles,b,x,its,ierr)
      call  SLESDestroy(sles,ierr)
```

beginner

solvers: linear



LANS

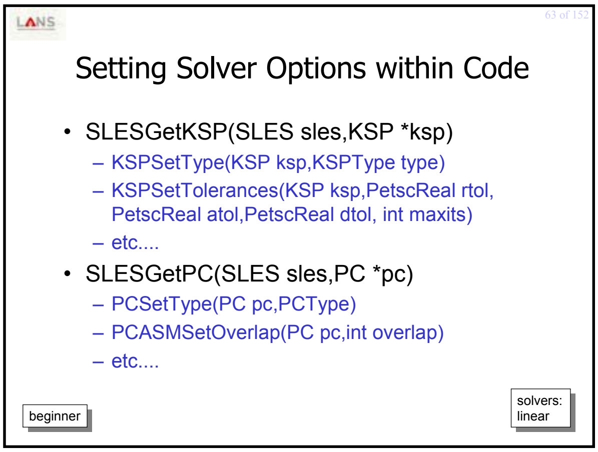
62 of 152

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



LANS

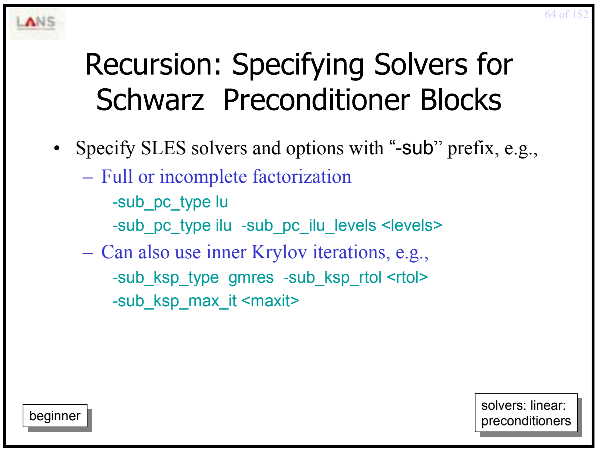
63 of 152

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....

beginner

solvers: linear



LANS

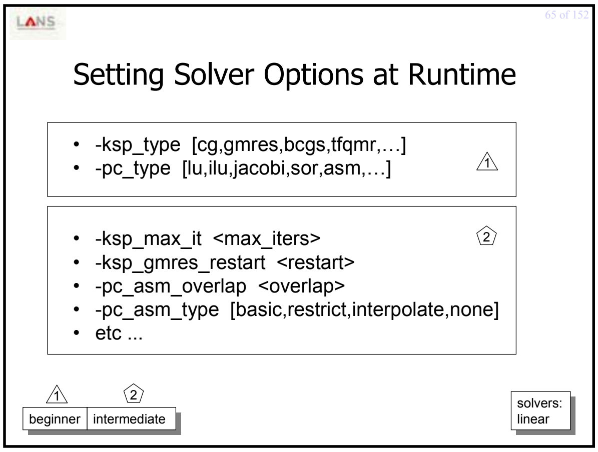
64 of 152

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>

beginner

solvers: linear: preconditioners



LANS

65 of 152

Setting Solver Options at Runtime

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

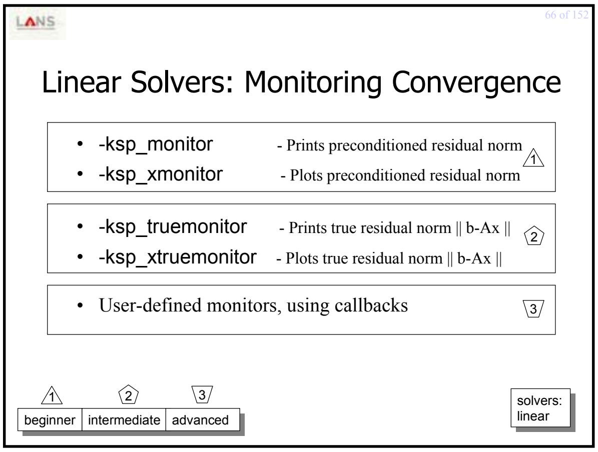
1

2

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

1 beginner | 2 intermediate

solvers: linear



LANS

66 of 152

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 beginner | 2 intermediate | 3 advanced

solvers: linear

Helmholtz: Scalability

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

Procs	Iterations	Time (Sec)	Speedup
1	221	163.01	-
2	222	81.06	2.0
4	224	37.36	4.4
8	228	19.49	8.4
16	229	10.85	15.0
32	230	6.37	25.6

beginner

solvers: linear

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

beginner

solvers: linear

SLES: 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 beginner ② intermediate

solvers: linear: preconditioners

SLES: Review of Selected Krylov Method Options

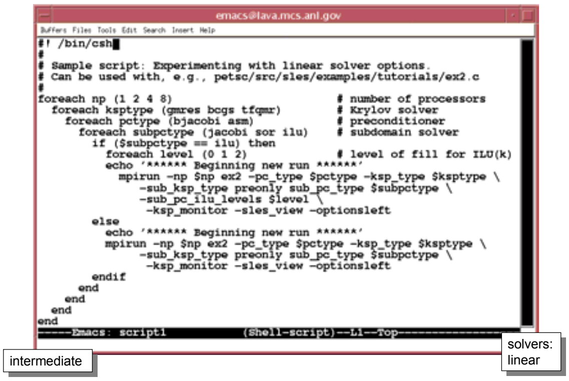
Functionality	Procedural Interface	Runtime Option
Set Krylov method	KSPSetType( )	-ksp_type [cg,gmres,bcgs, tfqmr,cgs,…] △1
Set monitoring routine	KSPSetMonitor( )	-ksp_monitor, –ksp_xmonitor, -ksp_truemonitor, -ksp_xtruemonitor
Set convergence tolerances	KSPSetTolerances( )	② -ksp_rtol <rt> -ksp_atol <at> -ksp_max_its <its>
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 beginner ② intermediate

solvers: linear: Krylov methods

SLES: Runtime Script Example

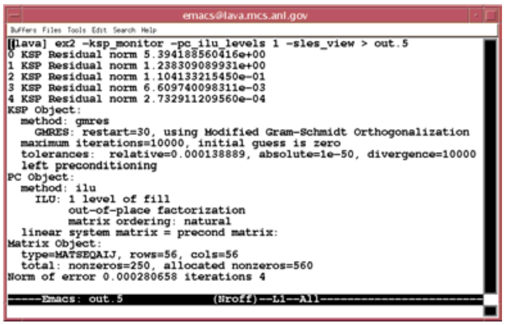


```
#! /bin/csh
#
# Sample script: Experimenting with linear solver options.
# Can be used with, e.g., petsc/src/sles/examples/tutorials/ex2.c
#
foreach np (1 2 4 8)                          # number of processors
  foreach ksptype (gmres bcgs tfqmr)          # Krylov solver
    foreach pctype (bjacobi asm)              # preconditioner
      foreach subpctype (jacobi sor ilu)      # subdomain solver
        if ($subpctype == ilu) then
          foreach level (0 1 2)               # level of fill for ILU(k)
          echo '****** Beginning new run ******'
            mpirun -np $np ex2 -pc_type $pctype -ksp_type $ksptype \
               -sub_ksp_type preonly sub_pc_type $subpctype \
               -sub_pc_ilu_levels $level \
                -ksp_monitor -sles_view -optionsleft
        else
          echo '****** Beginning new run ******'
          mpirun -np $np ex2 -pc_type $pctype -ksp_type $ksptype \
               -sub_ksp_type preonly sub_pc_type $subpctype \
                -ksp_monitor -sles_view -optionsleft
        endif
      end
    end
  end
end
-----Emacs: script1          (Shell-script)--L1--Top---------------
```

intermediate

solvers: linear

Viewing SLES Runtime Options



```
[lava] ex2 -ksp_monitor -pc_ilu_levels 1 -sles_view > out.5
0 KSP Residual norm 5.394188560416e+00
1 KSP Residual norm 1.238309089931e+00
2 KSP Residual norm 1.104133215450e-01
3 KSP Residual norm 6.609740098311e-03
4 KSP Residual norm 2.732911209560e-04
KSP Object:
  method: gmres
    GMRES: restart=30, using Modified Gram-Schmidt Orthogonalization
  maximum iterations=10000, initial guess is zero
  tolerances:  relative=0.000138889, absolute=1e-50, divergence=10000
  left preconditioning
PC Object:
  method: ilu
    ILU: 1 level of fill
         out-of-place factorization
         matrix ordering: natural
  linear system matrix = precond matrix:
Matrix Object:
  type=MATSEQAIJ, rows=56, cols=56
  total: nonzeros=250, allocated nonzeros=560
Norm of error 0.000280658 iterations 4

-----Emacs: out.5             (Nroff)--L1--All-------------------------
```

intermediate

solvers: linear

SLES: Example Programs

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

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

△1

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

⬠2

- ex15.c - setting a user-defined preconditioner

▽3

And many more examples ...

△1 beginner | ⬠2 intermediate | ▽3 advanced

Ⓔ - on-line exercise

solvers: linear

Profiling and Performance Tuning

Profiling:

- beginner • Integrated profiling using -log_summary
- intermediate • User-defined events
- intermediate • Profiling by stages of an application

Performance Tuning:

- intermediate • Matrix optimizations
- intermediate • Application optimizations
- advanced • Algorithmic tuning

tutorial outline: profiling and performance tuning

Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- All PETSc events are logged if compiled with -DPETSC_LOG (default); can also profile application code segments
- Print summary data with option: -log_summary
- Print redundant information from PETSc routines: -log_info
- Print the trace of the functions called: -log_trace

beginner

profiling and performance tuning

User-defined Events

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,”User event
   name”, “eventColor”);
PetscLogEventBegin(USER_EVENT,0,0,0,0);
[ code to monitor]
PetscLogFlops(user_evnet_flops);
PetscLogEvent End(USER_EVENT,0,0,0,0);
```

intermediate

profiling and performance tuning

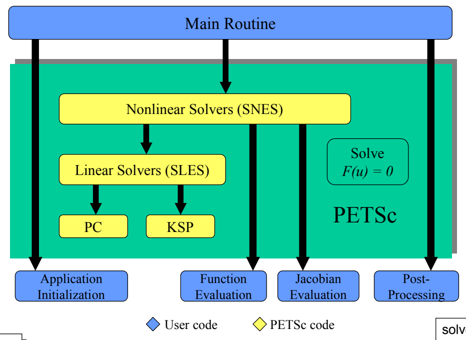
Nonlinear Solvers (SNES)

SNES: Scalable Nonlinear Equations Solvers

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

tutorial outline: solvers: nonlinear

Nonlinear PDE Solution



beginner

solvers: nonlinear

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)

beginner

solvers: nonlinear

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

beginner

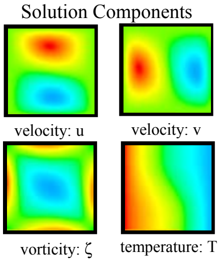
solvers: nonlinear

Sample Nonlinear Application:
Driven Cavity Problem

- Velocity-vorticity formulation
- Flow driven by lid and/or bouyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- source code:

petsc/src/snes/examples/tutorials/ex19.c

Solution Components



Application code author: D. E. Keyes

beginner

solvers: nonlinear

Basic Nonlinear Solver Code (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,PETSC_DECIDE,PETSC_DECIDE,n,n,&J);
VecCreate(MPI_COMM_WORLD,PETSC_DECIDE,n,&x);
VecDuplicate(x,&F);

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

beginner

solvers: nonlinear

Basic Nonlinear Solver Code (Fortran)

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

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

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

beginner

solvers: nonlinear

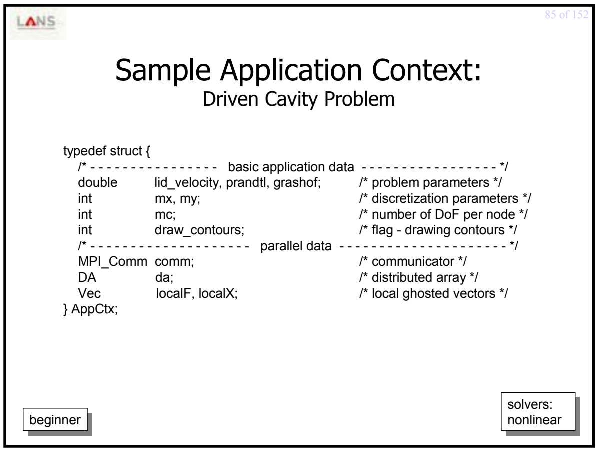
Solvers Based on Callbacks

- 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.

important concept

beginner

solvers: nonlinear



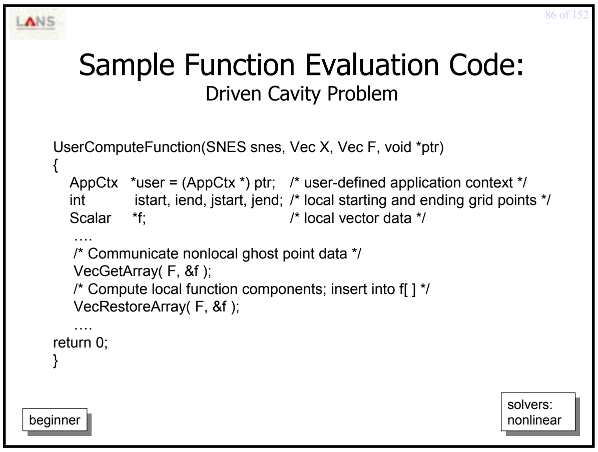
Sample Application Context:

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;
```

beginner

solvers:
nonlinear



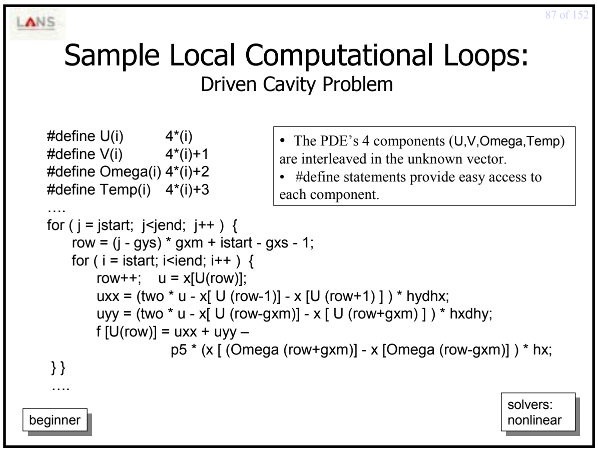
Sample Function Evaluation Code:

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;
}
```

beginner

solvers:
nonlinear



Sample Local Computational Loops:

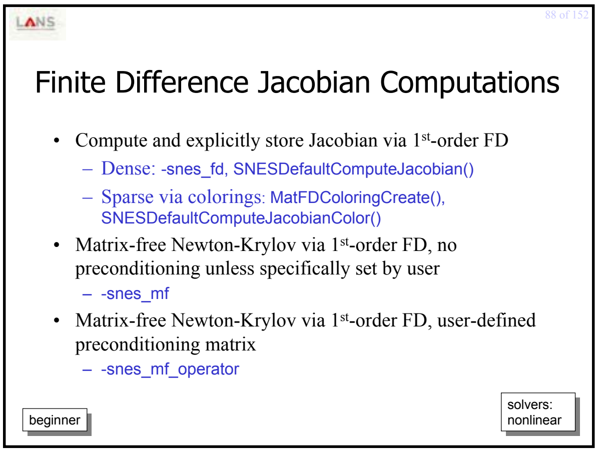
Driven Cavity Problem

- The PDE's 4 components (U,V,Omega,Temp) are interleaved in the unknown vector.
- #define statements provide easy access to each component.

```
#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) ] ) * hydhx;
            uyy = (two * u - x[ U (row-gxm)] - x [ U (row+gxm) ] ) * hxdhy;
            f [U(row)] = uxx + uyy –
                              p5 * (x [ (Omega (row+gxm)] - x [Omega (row-gxm)] ) * hx;
 } }
  ….
```

beginner

solvers:
nonlinear

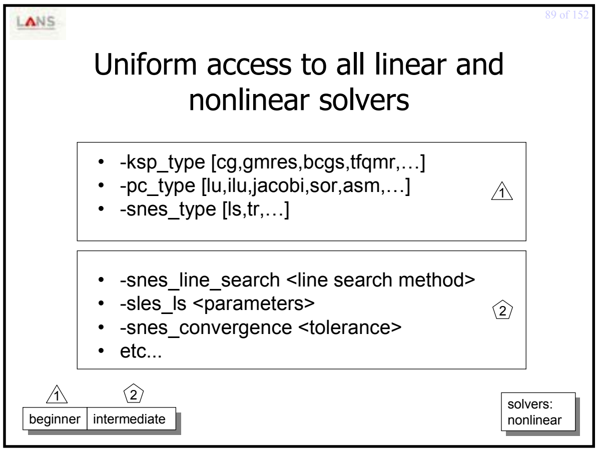


Finite Difference Jacobian Computations

- 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

beginner

solvers:
nonlinear



Uniform access to all linear and nonlinear solvers

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

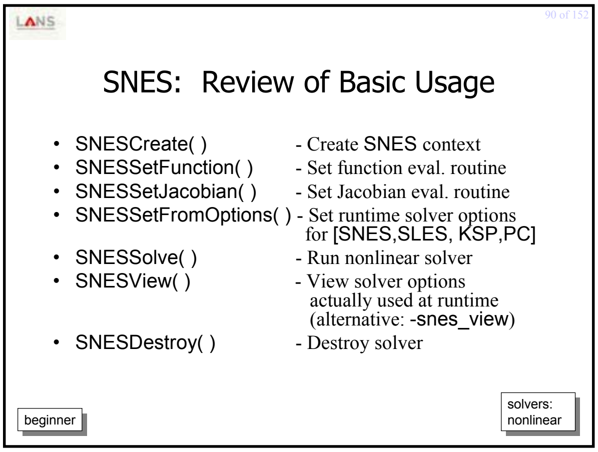
1

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

2

1 beginner | 2 intermediate

solvers:
nonlinear

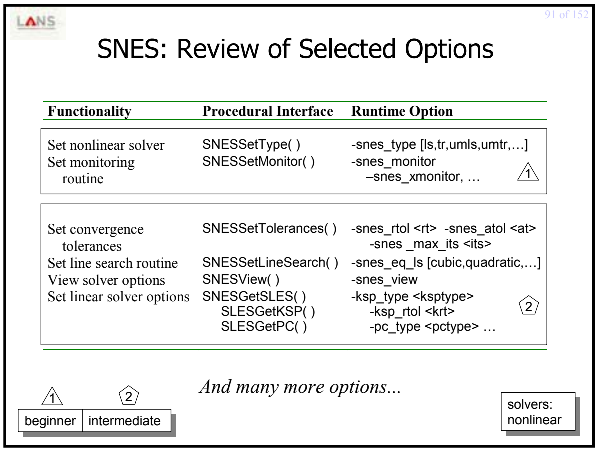


SNES: Review of Basic Usage

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

beginner

solvers:
nonlinear



SNES: Review of Selected Options

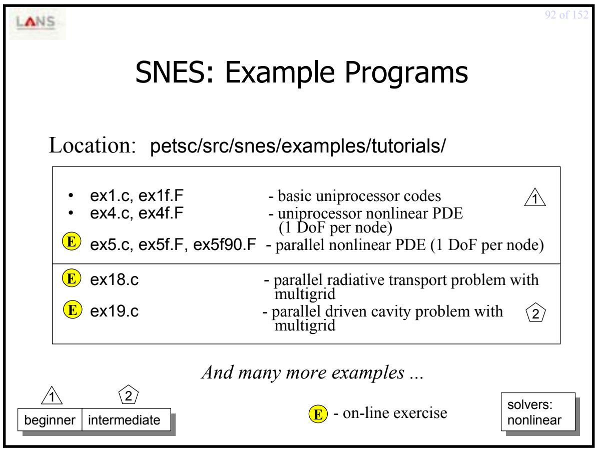
Functionality	Procedural Interface	Runtime Option
Set nonlinear solver	SNESSetType( )	-snes_type [ls,tr,umls,umtr,…]
Set monitoring routine	SNESSetMonitor( )	-snes_monitor –snes_xmonitor, … △1
Set convergence tolerances	SNESSetTolerances( )	-snes_rtol <rt> -snes_atol <at> -snes _max_its <its>
Set line search routine	SNESSetLineSearch( )	-snes_eq_ls [cubic,quadratic,…]
View solver options	SNESView( )	-snes_view

Set linear solver options	SNESGetSLES( )
	SLESGetKSP( )
	SLESGetPC( )
		-ksp_type <ksptype>
		-ksp_rtol <krt>
		-pc_type <pctype> … ⬠2

And many more options...

△1 beginner | ⬠2 intermediate

solvers: nonlinear



SNES: Example Programs

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

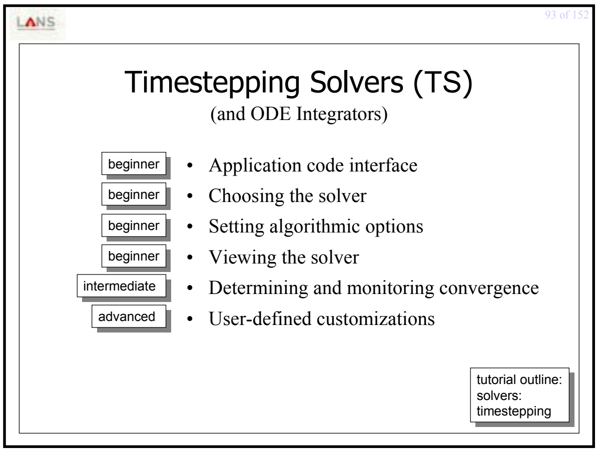
• ex1.c, ex1f.F	- basic uniprocessor codes △1
• ex4.c, ex4f.F	- uniprocessor nonlinear PDE (1 DoF per node)
Ⓔ ex5.c, ex5f.F, ex5f90.F	- parallel nonlinear PDE (1 DoF per node)
Ⓔ ex18.c	- parallel radiative transport problem with multigrid
Ⓔ ex19.c	- parallel driven cavity problem with multigrid ⬠2

And many more examples ...

△1 beginner | ⬠2 intermediate

Ⓔ - on-line exercise

solvers: nonlinear

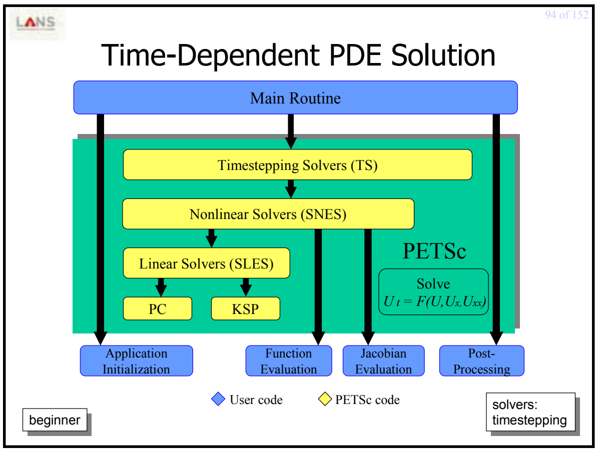


Timestepping Solvers (TS)

(and ODE Integrators)

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

tutorial outline: solvers: timestepping



Time-Dependent PDE Solution

Main Routine

Timestepping Solvers (TS)

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

PC

KSP

PETSc

Solve

$U_t = F(U,U_x,U_{xx})$

Application Initialization

Function Evaluation

Jacobian Evaluation

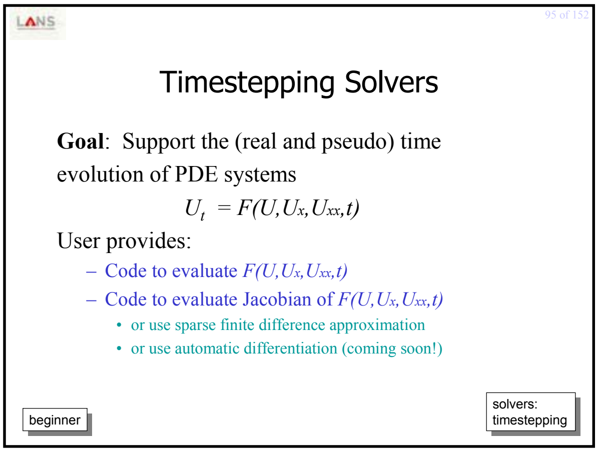
Post-Processing

🔷 User code

🔶 PETSc code

beginner

solvers: timestepping



Timestepping Solvers

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

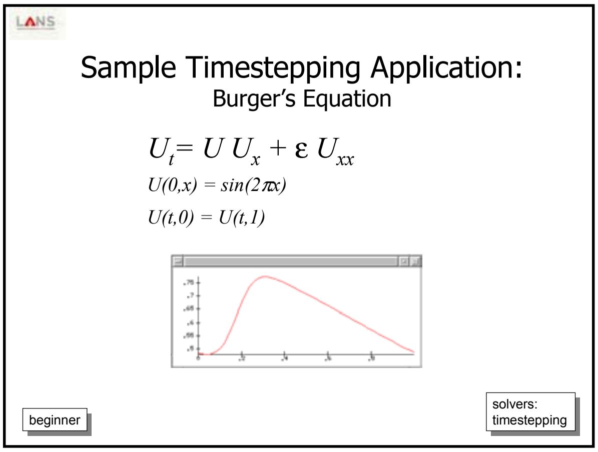
$$U_t = F(U,U_x,U_{xx},t)$$

User provides:

- Code to evaluate $F(U,U_x,U_{xx},t)$
- Code to evaluate Jacobian of $F(U,U_x,U_{xx},t)$
  - or use sparse finite difference approximation
  - or use automatic differentiation (coming soon!)

beginner

solvers: timestepping



Sample Timestepping Application:

Burger's Equation

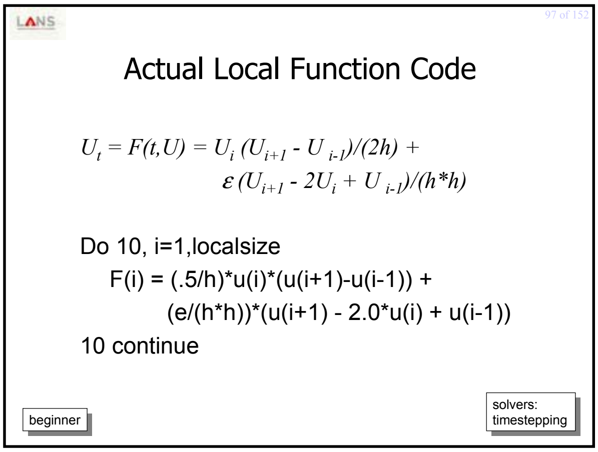
$$U_t = U\,U_x + \varepsilon\,U_{xx}$$

$U(0,x) = sin(2\pi x)$

$U(t,0) = U(t,1)$

beginner

solvers: timestepping



97 of 152

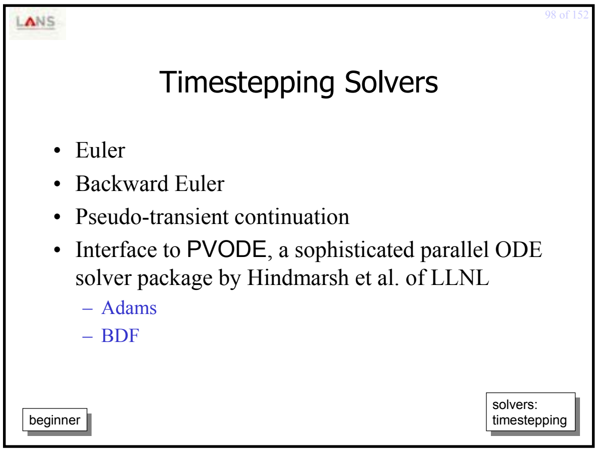
Actual Local Function Code

$$U_t = F(t,U) = U_i\,(U_{i+1} - U_{i-1})/(2h) + \varepsilon\,(U_{i+1} - 2U_i + U_{i-1})/(h*h)$$

```
Do 10, i=1,localsize
   F(i) = (.5/h)*u(i)*(u(i+1)-u(i-1)) +
          (e/(h*h))*(u(i+1) - 2.0*u(i) + u(i-1))
10 continue
```

beginner

solvers: timestepping



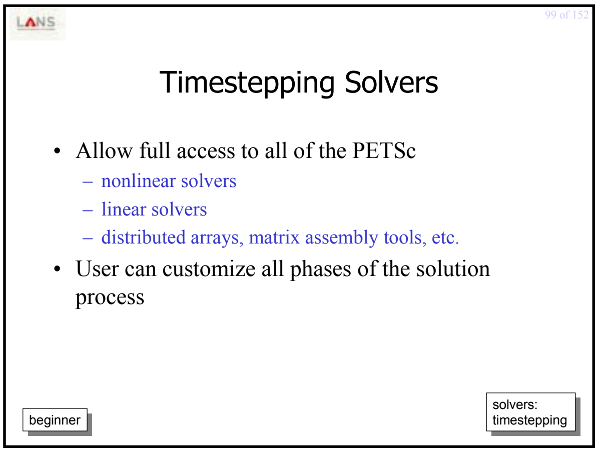
98 of 152

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

beginner

solvers: timestepping



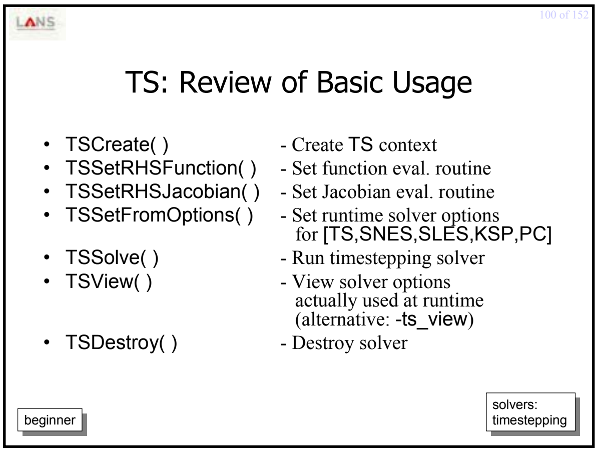
99 of 152

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

beginner

solvers: timestepping



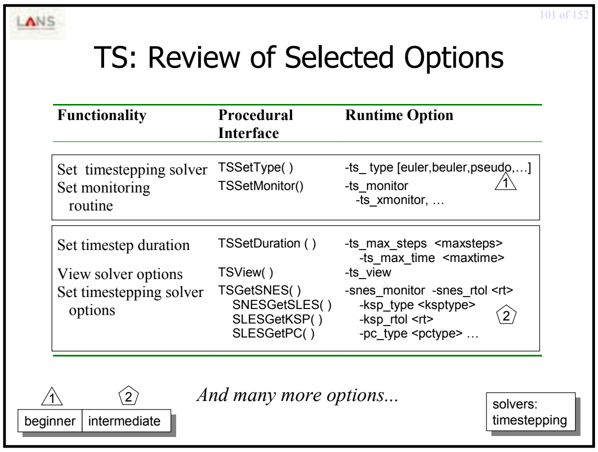
100 of 152

TS: Review of Basic Usage

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

beginner

solvers: timestepping



101 of 152

TS: Review of Selected Options

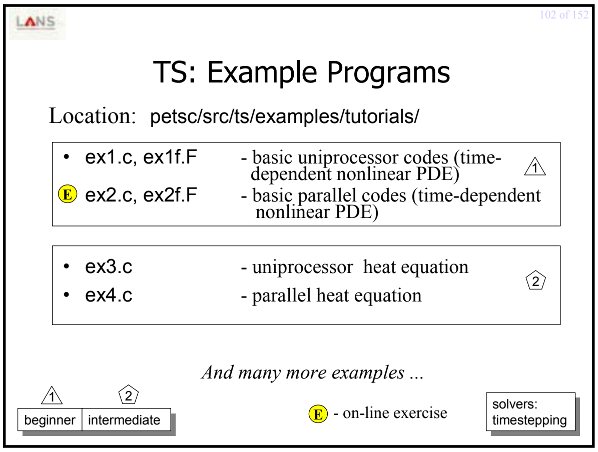
Functionality	Procedural Interface	Runtime Option
Set timestepping solver	TSSetType( )	-ts_ type [euler,beuler,pseudo,…]
Set monitoring routine	TSSetMonitor()	-ts_monitor -ts_xmonitor, … △1
Set timestep duration	TSSetDuration ( )	-ts_max_steps <maxsteps> -ts_max_time <maxtime>
View solver options	TSView( )	-ts_view

Set timestepping solver options	TSGetSNES( )
	SNESGetSLES( )
	SLESGetKSP( )
	SLESGetPC( )
		-snes_monitor -snes_rtol <rt>
		-ksp_type <ksptype>
		-ksp_rtol <rt>
		-pc_type <pctype> … ⬠2

And many more options...

△1 beginner ⬠2 intermediate

solvers: timestepping



102 of 152

TS: Example Programs

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

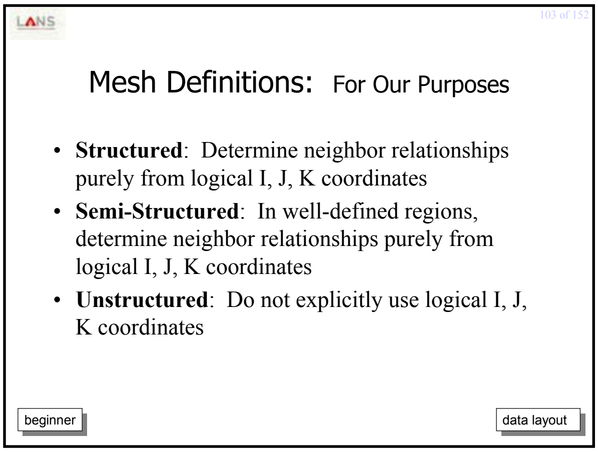
• ex1.c, ex1f.F	- basic uniprocessor codes (time-dependent nonlinear PDE)	△1
Ⓔ ex2.c, ex2f.F	- basic parallel codes (time-dependent nonlinear PDE)	
• ex3.c	- uniprocessor heat equation	⬠2
• ex4.c	- parallel heat equation	

And many more examples ...

△1 beginner ⬠2 intermediate

Ⓔ - on-line exercise

solvers: timestepping

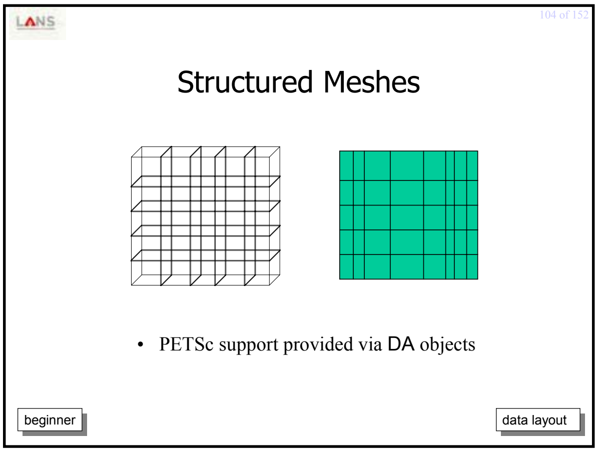


Mesh Definitions: For Our Purposes

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

beginner

data layout

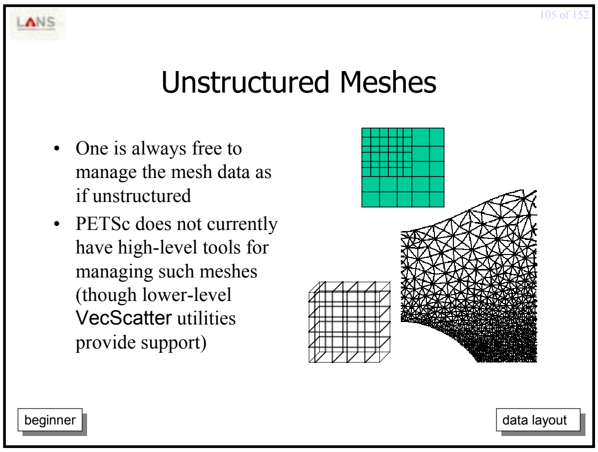


Structured Meshes

- PETSc support provided via DA objects

beginner

data layout

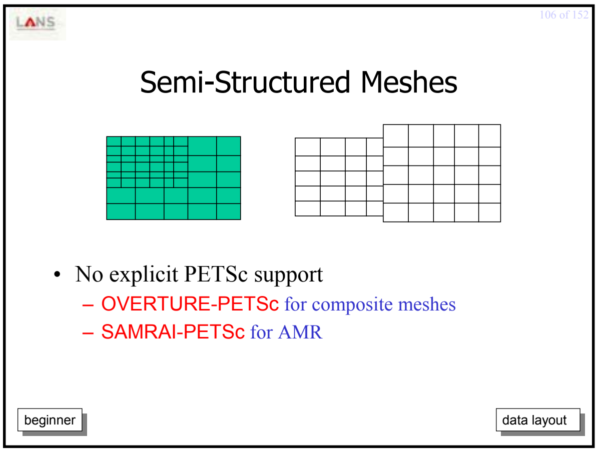


Unstructured Meshes

- 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)

beginner

data layout

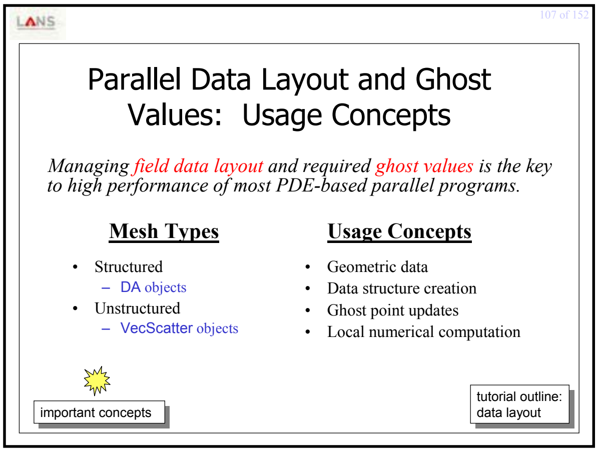


Semi-Structured Meshes

- No explicit PETSc support
  - OVERTURE-PETSc for composite meshes
  - SAMRAI-PETSc for AMR

beginner

data layout



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.

Mesh Types

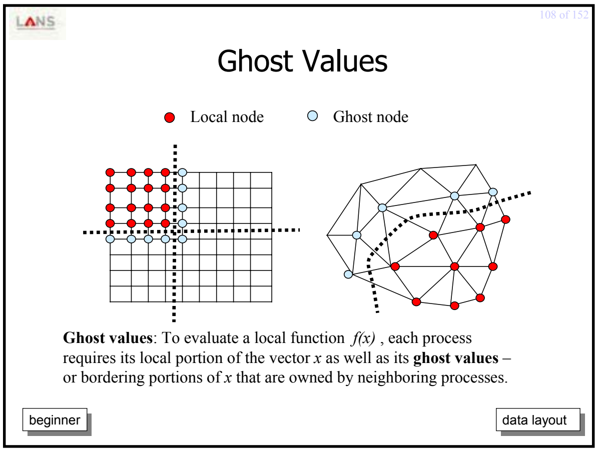
- Structured
  - DA objects
- Unstructured
  - VecScatter objects

Usage Concepts

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

important concepts

tutorial outline: data layout



Ghost Values

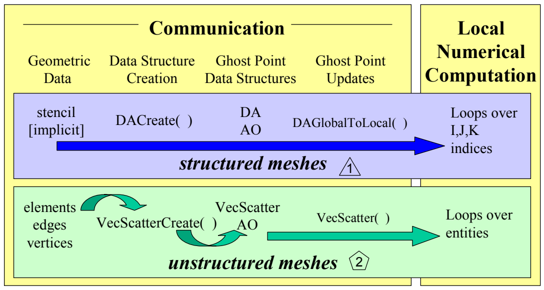
● Local node ○ Ghost node

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.

beginner

data layout

Communication and Physical Discretization



1 2

beginner intermediate

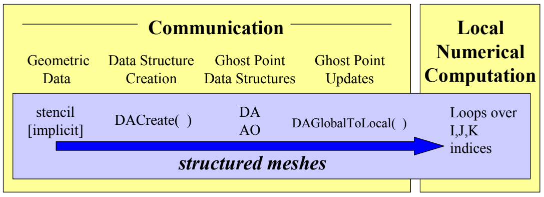
data layout

DA: Parallel Data Layout and Ghost Values for Structured Meshes

beginner	• Local and global indices
beginner	• Local and global vectors
beginner	• DA creation
intermediate	• Ghost point updates
intermediate	• Viewing

tutorial outline:
data layout:
distributed arrays

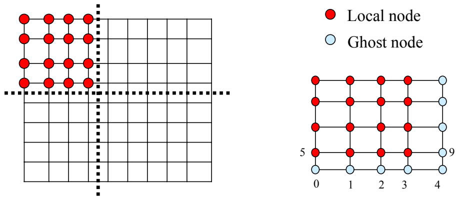
Communication and Physical Discretization: Structured Meshes



beginner

data layout:
distributed arrays

Global and Local Representations



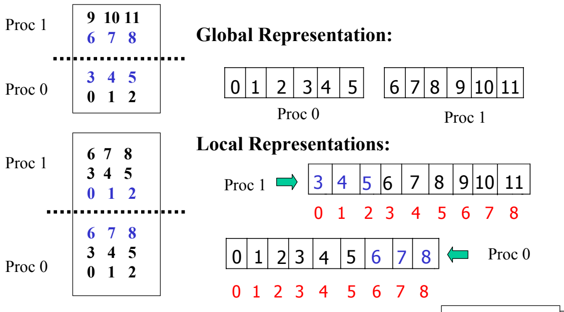
Global: each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

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

beginner

data layout:
distributed arrays

Global and Local Representations (cont.)



beginner

data layout:
distributed arrays

Logically Regular Meshes

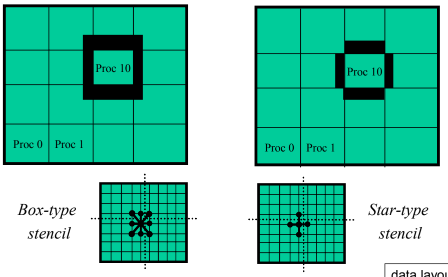
- DA - Distributed Array: object containing information about vector layout across the processes and communication of ghost values
- Form a DA
  - DACreateXX(….,DA *)
- Update ghostpoints
  - DAGlobalToLocalBegin(DA,…)
  - DAGlobalToLocalEnd(DA,…)

beginner

data layout:
distributed arrays

Distributed Arrays

Data layout and ghost values



beginner

data layout: 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)

beginner

data layout: distributed arrays

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
- …

beginner

data layout: distributed arrays

DACreate2d(…,*DA)

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

And similarly for DACreate3d()

beginner

data layout: distributed arrays

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,…)

beginner

data layout: distributed arrays

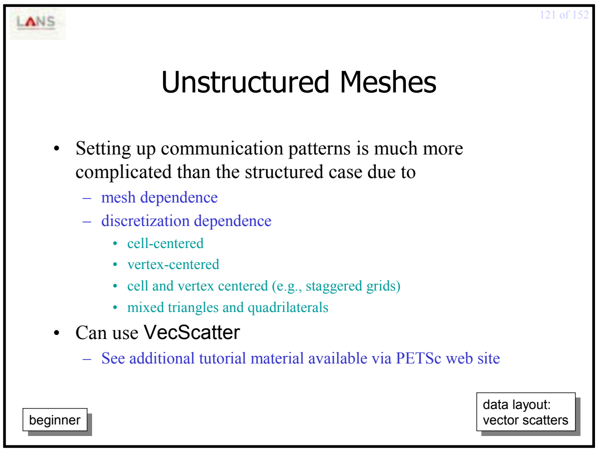
Ghost Point Scatters:

Burger's Equation Example

```
      call DAGlobalToLocalBegin(da,u_global,INSERT_VALUES,u,ierr)
      call DAGlobalToLocalEnd(da,u_global,INSERT_VALUES,u,ierr)

C  Do local computations (here u and f are local vectors)
      Do 10, i=1,localsize
         f(i) = (.5/h)*u(i)*(u(i+1)-u(i-1)) +
                    (e/(h*h))*(u(i+1) - 2.0*u(i) + u(i-1))
10  continue

      call DALocalToGlobal(da,f,INSERT_VALUES,f_global,ierr)
```

beginner

data layout: distributed arrays

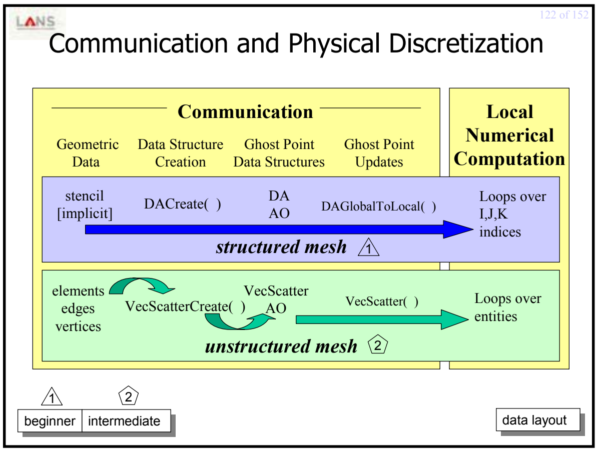


Unstructured Meshes

- 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

beginner

data layout: vector scatters



Communication and Physical Discretization

Communication

Geometric Data

Data Structure Creation

Ghost Point Data Structures

Ghost Point Updates

Local Numerical Computation

stencil [implicit]

DACreate( )

DA AO

DAGlobalToLocal( )

Loops over I,J,K indices

structured mesh △1

elements edges vertices

VecScatterCreate( )

VecScatter AO

VecScatter( )

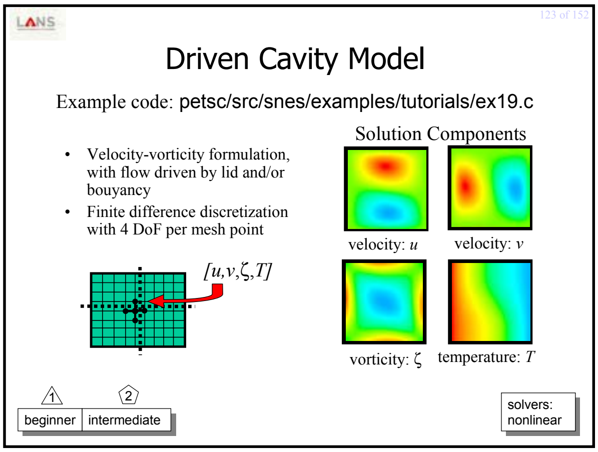
Loops over entities

unstructured mesh ⬠2

△1 beginner

⬠2 intermediate

data layout



Driven Cavity Model

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

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

[u,v,ζ,T]

Solution Components

velocity: *u*

velocity: *v*

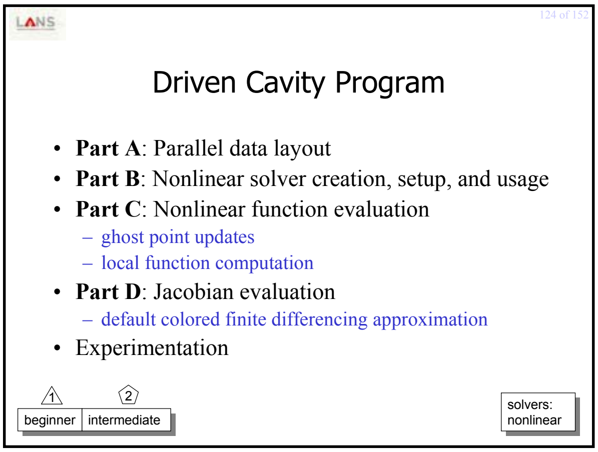
vorticity: ζ

temperature: *T*

△1 beginner

⬠2 intermediate

solvers: nonlinear



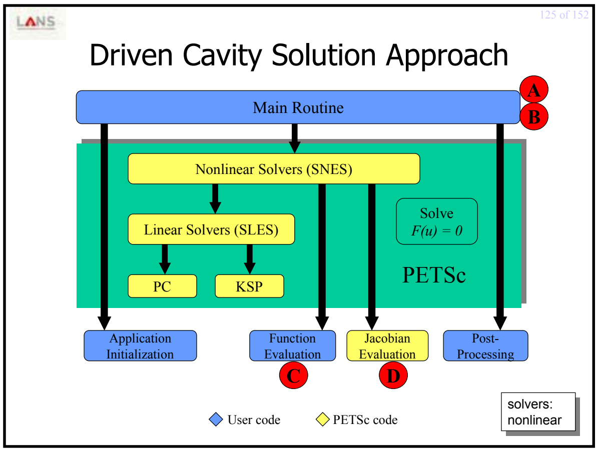
Driven Cavity Program

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

△1 beginner

⬠2 intermediate

solvers: nonlinear



Driven Cavity Solution Approach

Main Routine

A

B

Nonlinear Solvers (SNES)

Linear Solvers (SLES)

PC

KSP

Solve
F(u) = 0

PETSc

Application Initialization

Function Evaluation

C

Jacobian Evaluation

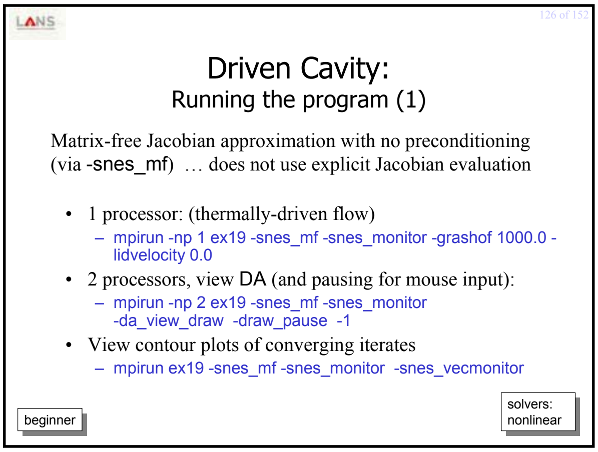
D

Post-Processing

🔷 User code

🔶 PETSc code

solvers: nonlinear



Driven Cavity:

Running the program (1)

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

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

beginner

solvers: nonlinear

Debugging and Error Handling

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

tutorial outline:
debugging and errors

Debugging

Support for parallel debugging

- -start_in_debugger [gdb,dbx,noxterm]
- -on_error_attach_debugger [gb,dbx,noxterm]
- -on_error_abort
- -debugger_nodes 0,1
- -display machinename:0.0

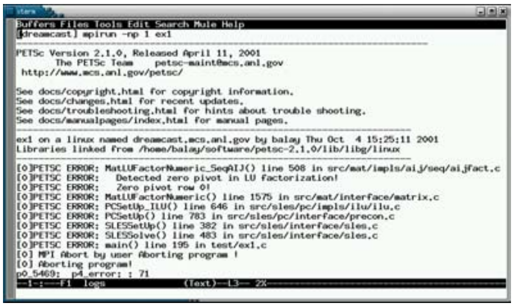
When debugging, it is often useful to place a breakpoint in the function PetscError( ).

beginner

debugging and errors

Sample Error Traceback

Breakdown in ILU factorization due to a zero pivot

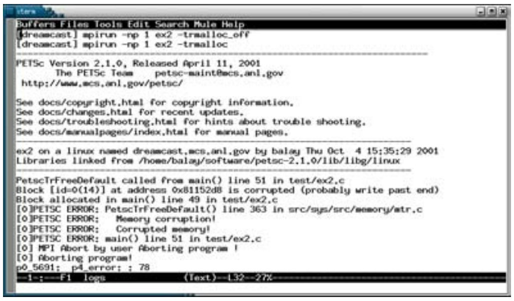


```
[dreamcast] mpirun -np 1 ex1
--------------------------------------------------------------------------
PETSc Version 2.1.0, Released April 11, 2001
       The PETSc Team    petsc-maint@mcs.anl.gov
 http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.
-----------------------------------------------------------------------
ex1 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:25:11 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux
-----------------------------------------------------------------------
[0]PETSC ERROR: MatLUFactorNumeric_SeqAIJ() line 508 in src/mat/impls/aij/seq/aijfact.c
[0]PETSC ERROR:   Detected zero pivot in LU factorization!
[0]PETSC ERROR:   Zero pivot row 0!
[0]PETSC ERROR: MatLUFactorNumeric() line 1575 in src/mat/interface/matrix.c
[0]PETSC ERROR: PCSetUp_ILU() line 646 in src/sles/pc/impls/ilu/ilu.c
[0]PETSC ERROR: PCSetUp() line 783 in src/sles/pc/interface/precon.c
[0]PETSC ERROR: SLESSetUp() line 382 in src/sles/interface/sles.c
[0]PETSC ERROR: SLESSolve() line 483 in src/sles/interface/sles.c
[0]PETSC ERROR: main() line 195 in test/ex1.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_5469:  p4_error: : 71
--1-:---F1  logs          (Text)--L3-- 2%-----------------------------------------
```

beginner

debugging and errors

Sample Memory Corruption Error

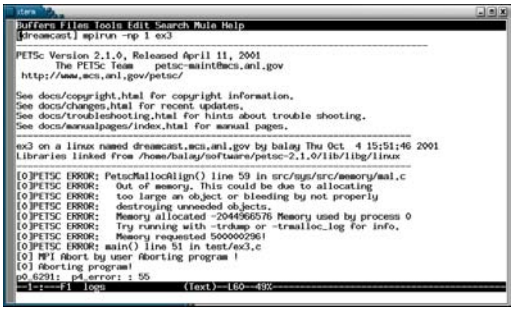


```
[dreamcast] mpirun -np 1 ex2 -trmalloc_off
[dreamcast] mpirun -np 1 ex2 -trmalloc
--------------------------------------------------------------------------
PETSc Version 2.1.0, Released April 11, 2001
       The PETSc Team    petsc-maint@mcs.anl.gov
 http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.
-----------------------------------------------------------------------
ex2 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:35:29 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux
-----------------------------------------------------------------------
PetscTrFreeDefault called from main() line 51 in test/ex2.c
Block [id=0(14)] at address 0x81152d8 is corrupted (probably write past end)
Block allocated in main() line 49 in test/ex2.c
[0]PETSC ERROR: PetscTrFreeDefault() line 363 in src/sys/src/memory/mtr.c
[0]PETSC ERROR:   Memory corruption!
[0]PETSC ERROR:   Corrupted memory!
[0]PETSC ERROR: main() line 51 in test/ex2.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_5691:  p4_error: : 78
--1-:---F1  logs          (Text)--L32--27%----------------------------------------
```

beginner

debugging and errors

Sample Out-of-Memory Error

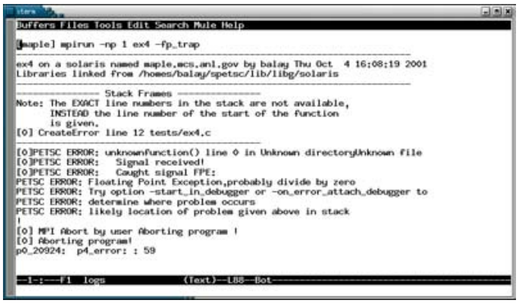


```
[dreamcast] mpirun -np 1 ex3
--------------------------------------------------------------------------
PETSc Version 2.1.0, Released April 11, 2001
       The PETSc Team    petsc-maint@mcs.anl.gov
 http://www.mcs.anl.gov/petsc/

See docs/copyright.html for copyright information.
See docs/changes.html for recent updates.
See docs/troubleshooting.html for hints about trouble shooting.
See docs/manualpages/index.html for manual pages.
-----------------------------------------------------------------------
ex3 on a linux named dreamcast.mcs.anl.gov by balay Thu Oct  4 15:51:46 2001
Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux
-----------------------------------------------------------------------
[0]PETSC ERROR: PetscMallocAlign() line 59 in src/sys/src/memory/mal.c
[0]PETSC ERROR:   Out of memory. This could be due to allocating
[0]PETSC ERROR:   too large an object or bleeding by not properly
[0]PETSC ERROR:   destroying unneeded objects.
[0]PETSC ERROR:   Memory allocated -2044966576 Memory used by process 0
[0]PETSC ERROR:   Try running with -trdump or -trmalloc_log for info.
[0]PETSC ERROR:   Memory requested 500000296!
[0]PETSC ERROR: main() line 51 in test/ex3.c
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_6291:  p4_error: : 55
--1-:---F1  logs          (Text)--L60--49%----------------------------------------
```

beginner

debugging and errors

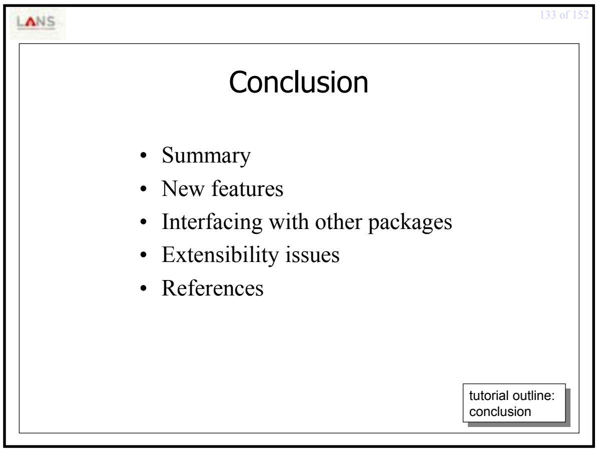
Sample Floating Point Error



```
[maple] mpirun -np 1 ex4 -fp_trap
-----------------------------------------------------------------------
ex4 on a solaris named maple.mcs.anl.gov by balay Thu Oct  4 16:08:19 2001
Libraries linked from /homes/balay/spetsc/lib/libg/solaris
-----------------------------------------------------------------------
--------------- Stack Frames ---------------
Note: The EXACT line numbers in the stack are not available,
      INSTEAD the line number of the start of the function
      is given.
[0] CreateError line 12 tests/ex4.c
--------------------------------------------
[0]PETSC ERROR: unknownfunction() line 0 in Unknown directoryUnknown file
[0]PETSC ERROR:   Signal received!
[0]PETSC ERROR:   Caught signal FPE:
PETSC ERROR: Floating Point Exception,probably divide by zero
PETSC ERROR: Try option -start_in_debugger or -on_error_attach_debugger to
PETSC ERROR: determine where problem occurs
PETSC ERROR: likely location of problem given above in stack
!
[0] MPI Abort by user Aborting program !
[0] Aborting program!
p0_20924:  p4_error: : 59


--1-:---F1  logs          (Text)--L88--Bot----------------------------------------
```

beginner

debugging and errors

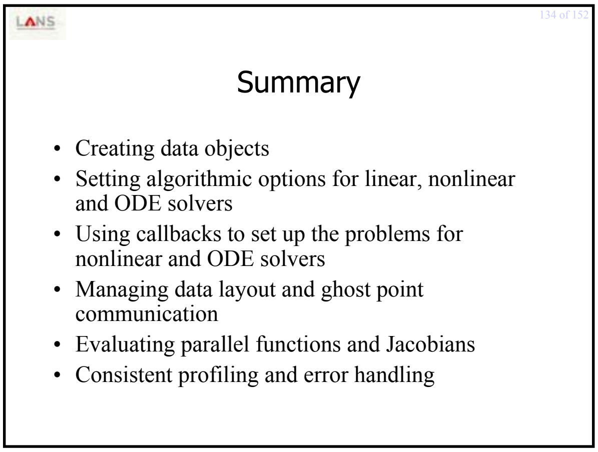


133 of 152

Conclusion

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

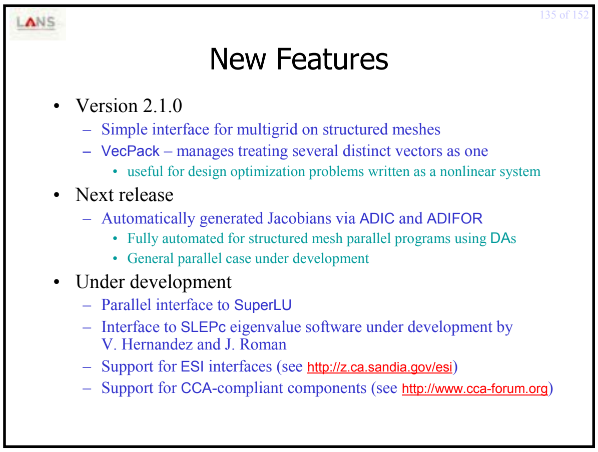
tutorial outline: conclusion



134 of 152

Summary

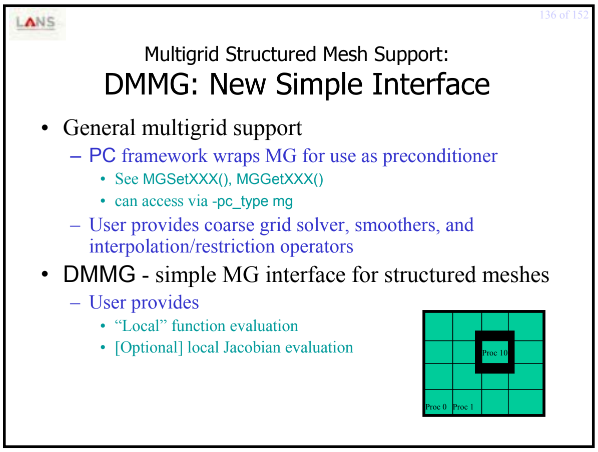
- Creating data objects
- Setting algorithmic options for linear, nonlinear and ODE solvers
- Using callbacks to set up the problems for nonlinear and ODE solvers
- Managing data layout and ghost point communication
- Evaluating parallel functions and Jacobians
- Consistent profiling and error handling



135 of 152

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 ESI interfaces (see http://z.ca.sandia.gov/esi)
  - Support for CCA-compliant components (see http://www.cca-forum.org)



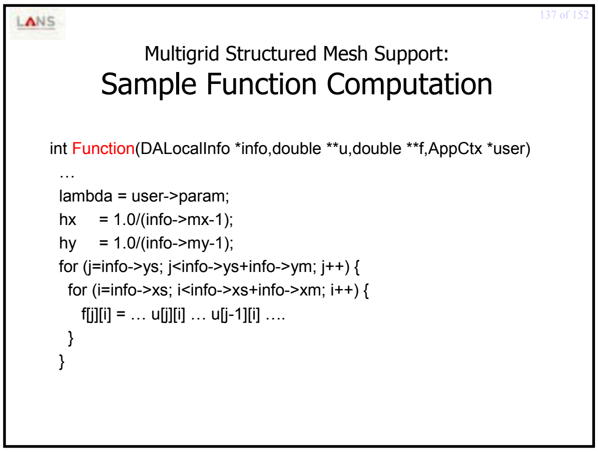
136 of 152

Multigrid Structured Mesh Support:

DMMG: New Simple Interface

- General multigrid support
  - PC framework wraps MG for use as preconditioner
    - 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

Proc 10
Proc 0
Proc 1

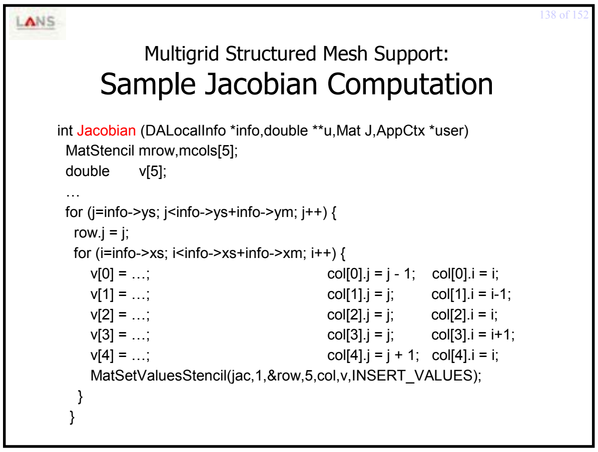


137 of 152

Multigrid Structured Mesh Support:

Sample Function Computation

```
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] ….
   }
 }
```

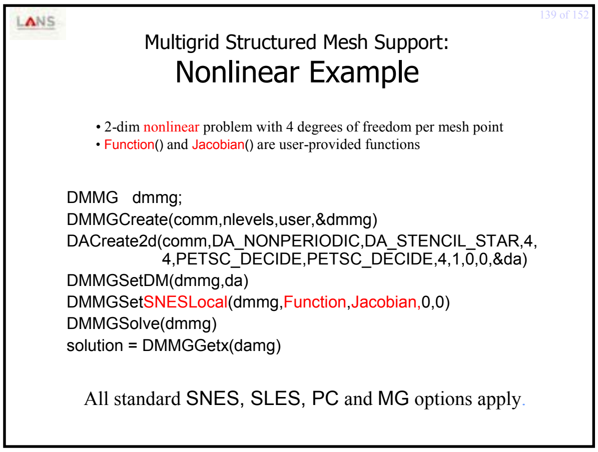


138 of 152

Multigrid Structured Mesh Support:

Sample Jacobian Computation

```
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;
      v[2] = …;                     col[2].j = j;       col[2].i = i;
      v[3] = …;                     col[3].j = j;       col[3].i = i+1;
      v[4] = …;                     col[4].j = j + 1;   col[4].i = i;
      MatSetValuesStencil(jac,1,&row,5,col,v,INSERT_VALUES);
    }
  }
```

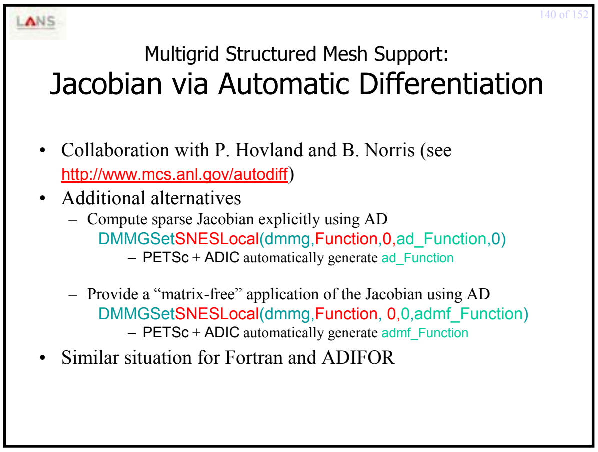


Multigrid Structured Mesh Support:
Nonlinear Example

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

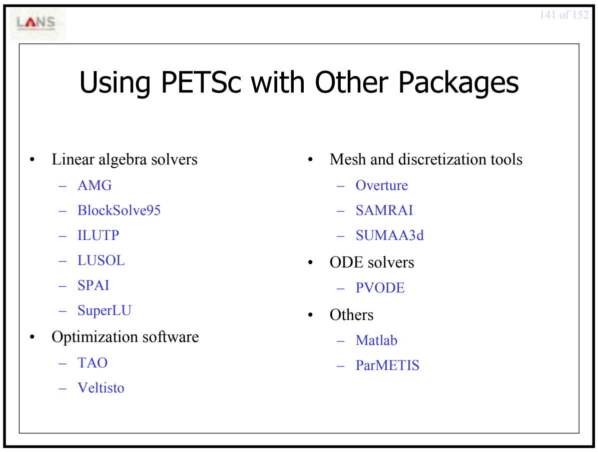
```
DMMG   dmmg;
DMMGCreate(comm,nlevels,user,&dmmg)
DACreate2d(comm,DA_NONPERIODIC,DA_STENCIL_STAR,4,
            4,PETSC_DECIDE,PETSC_DECIDE,4,1,0,0,&da)
DMMGSetDM(dmmg,da)
DMMGSetSNESLocal(dmmg,Function,Jacobian,0,0)
DMMGSolve(dmmg)
solution = DMMGGetx(damg)
```

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



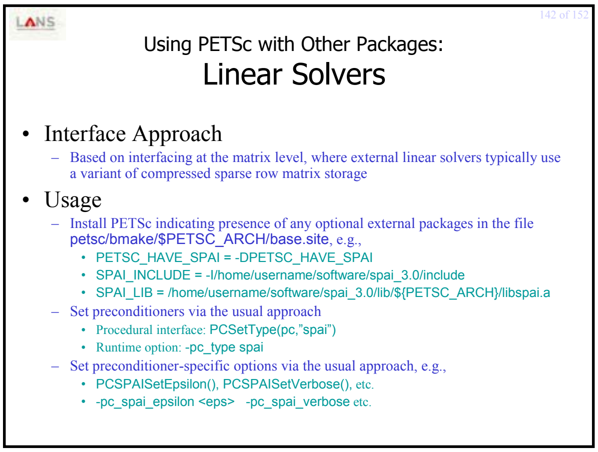
Multigrid Structured Mesh Support:
Jacobian via Automatic Differentiation

- 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,0,ad_Function,0)
    - PETSc + ADIC automatically generate ad_Function
  - Provide a "matrix-free" application of the Jacobian using AD
    DMMGSetSNESLocal(dmmg,Function, 0,0,admf_Function)
    - PETSc + ADIC automatically generate admf_Function
- Similar situation for Fortran and ADIFOR



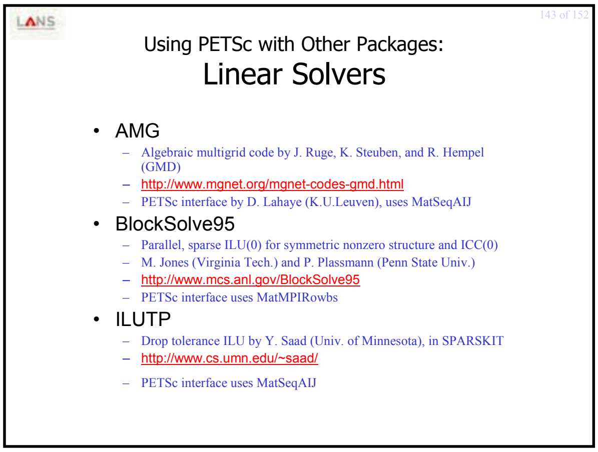
Using PETSc with Other Packages

- 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



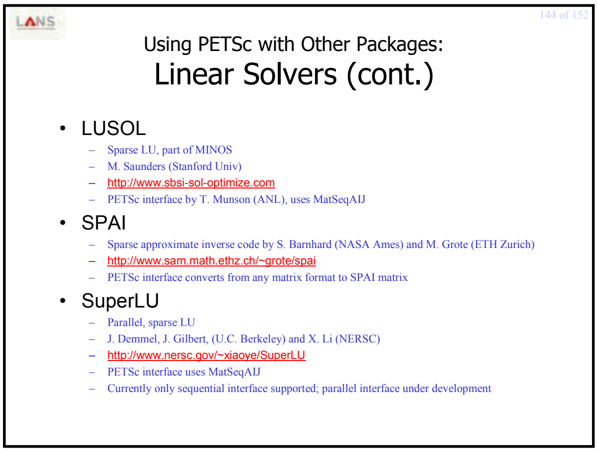
Using PETSc with Other Packages:
Linear Solvers

- 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
    - Procedural interface: PCSetType(pc,"spai")
    - Runtime option: -pc_type spai
  - Set preconditioner-specific options via the usual approach, e.g.,
    - PCSPAISetEpsilon(), PCSPAISetVerbose(), etc.
    - -pc_spai_epsilon <eps>   -pc_spai_verbose etc.



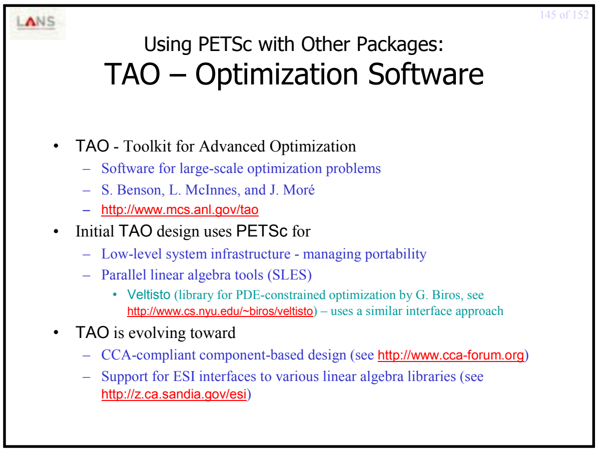
Using PETSc with Other Packages:
Linear Solvers

- 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



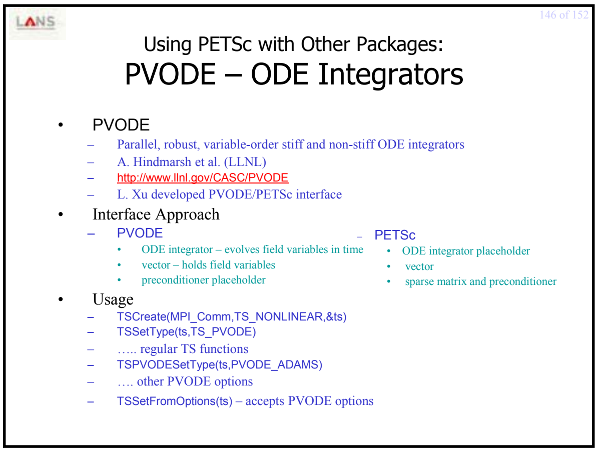
Using PETSc with Other Packages:
Linear Solvers (cont.)

- 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



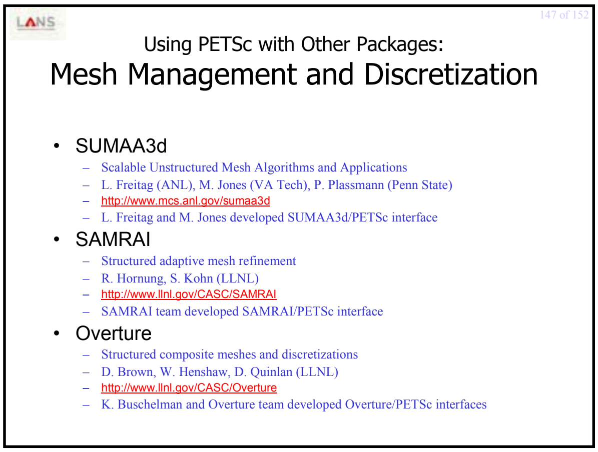
Using PETSc with Other Packages: TAO – Optimization Software

- 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 (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)



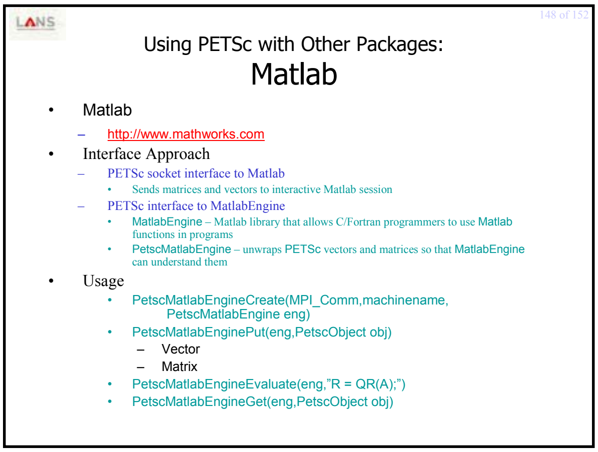
Using PETSc with Other Packages: PVODE – ODE Integrators

- 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



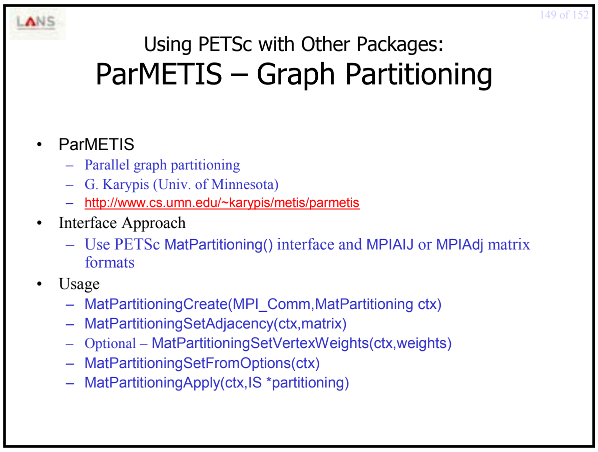
Using PETSc with Other Packages: Mesh Management and Discretization

- 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



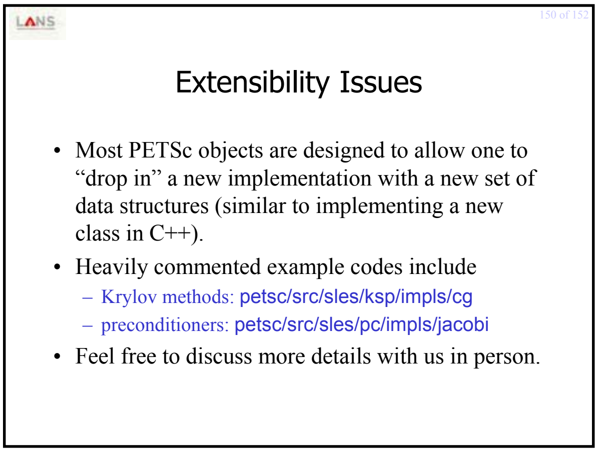
Using PETSc with Other Packages: Matlab

- 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)



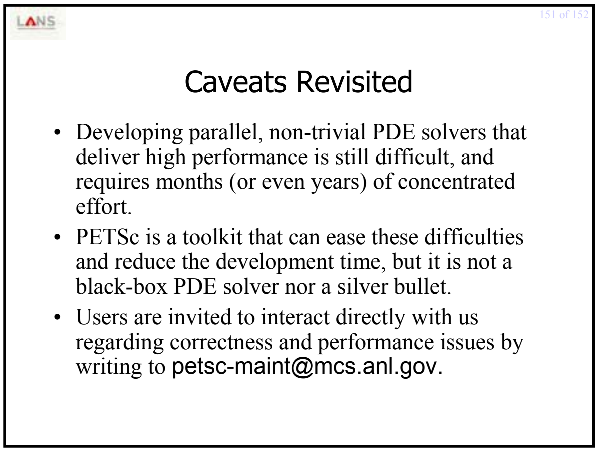
Using PETSc with Other Packages: ParMETIS – Graph Partitioning

- 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)



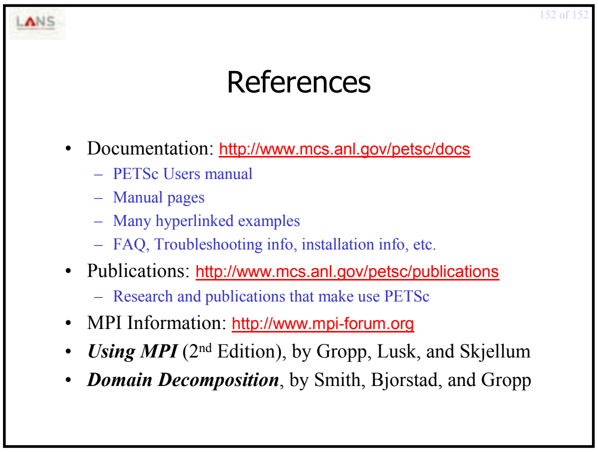
Extensibility Issues

- 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/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