PETSc and its Ongoing Research and Development

PETSc Team

Presented by Hong Zhang
CScADS Workshop on Libraries and Algorithms for Petascale Applications, Snowbird Utah
Aug. 1, 2007

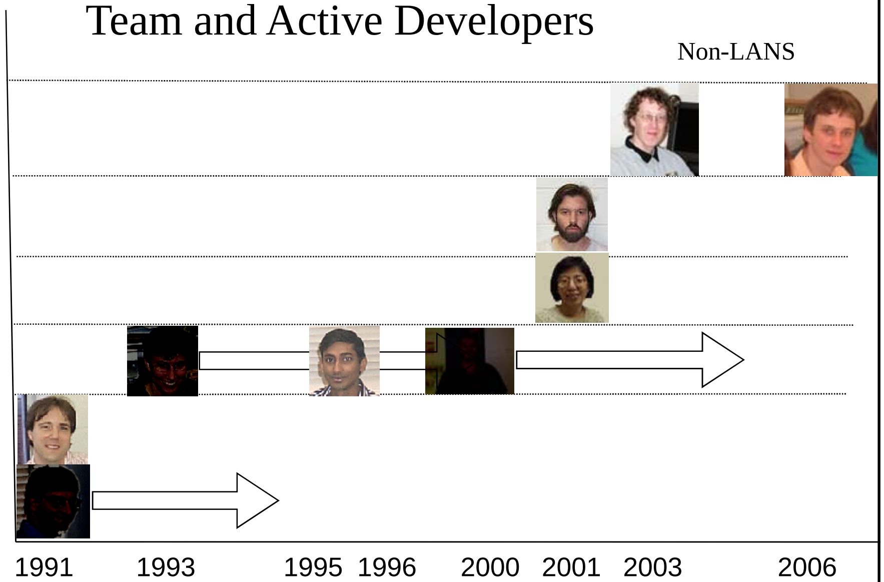
1

Outline

- Overview of PETSc
  - Linear solver interface: KSP
  - Nonlinear solver interface: SNES
  - Profiling and debugging
- Ongoing research and developments

2

Team and Active Developers



How did PETSc Originate?

PETSc was developed as a Platform for
Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms (which blur these boundaries)

Successfully transitioned from basic research to common community tool

- Applications of PETSc
- Nano-simulations (20)
- Biology/Medical(28)
- Cardiology
- Imaging and Surgery
- Fusion (10)
- Geosciences (20)
- Environmental/Subsurface Flow (26)
- Computational Fluid Dynamics (49)
- Wave propagation and the Helmholz equation (12)
- Optimization (7)
- Other Application Areas (68)
- Software packages that use or interface to PETSc (30)
- Software engineering (30)
- Algorithm analysis and design (48)

5

Who Uses PETSc?

- Computational Scientists
  - PyLith (TECTON), Underworld, Columbia group
- Algorithm Developers
  - Iterative methods and Preconditioning researchers
- Package Developers
  - SIPs, SLEPc, TAO, MagPar, StGermain, Dealll

6

The Role of PETSc

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

PETSc is a tool that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

-Barry Smith

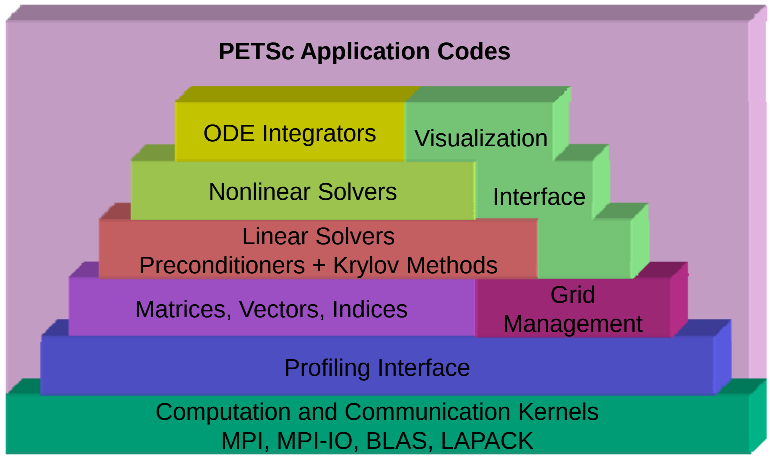
7

Features

- Many (parallel) vector/array operations
- Numerous (parallel) matrix formats and operations
- **Numerous** linear solvers
- Nonlinear solvers
- Limited ODE integrators
- Limited parallel grid/data management
- Common interface for most DOE solver software

8

Structure of PETSc



PETSc Structure

Interfaced Packages

1. LU (Sequential)
   - SuperLU (Demmel and Li, LBNL)
   - ESSL (IBM)
   - Matlab
   - LUSOL (from MINOS - Michael Saunders, Stanford)
   - LAPACK
   - PLAPACK (van de Geijn, UT Austin)
   - UMFPACK (Timothy A. Davis)
2. Parallel LU
   - SuperLU_DIST (Demmel and Li, LBNL)
   - SPOOLES (Ashcroft, Boeing, funded by ARPA)
   - MUMPS (European)
   - PLAPACK (van de Geijn, UT Austin)
3. Parallel Cholesky
   - DSCPACK (Raghavan, Penn. State)
   - SPOOLES (Ashcroft, Boeing, funded by ARPA)
   - PLAPACK (van de Geijn, UT Austin)

10

Interfaced Packages

4. XYTlib – parallel direct solver (Fischer and Tufo, ANL)
5. SPAI – Sparse approximate inverse (parallel)
   - Parasails (Chow, part of Hypre, LLNL)
   - SPAI 3.0 (Grote/Barnard)
6. Algebraic multigrid
   - Parallel BoomerAMG (part of Hypre, LLNL)
   - ML (part of Trilinos, SNL)
7. Parallel ICC(0) – BlockSolve95 (Jones and Plassman, ANL)
8. Parallel ILU
   - BlockSolve95 (Jones and Plassman, ANL)
   - PILUT (part of Hypre, LLNL)
   - EUCLID (Hysom – also part of Hypre, ODU/LLNL)
9. Sequential ILUDT (SPARSEKIT2- Y. Saad, U of MN)

11

Interfaced Packages

10. Parititioning
    - Parmetis
    - Chaco
    - Jostle
    - Party
    - Scotch
11. ODE integrators
    - Sundials (LLNL)
12. Eigenvalue solvers
    - BLOPEX (developed by Andrew Knyazev)

12

Child Packages of PETSc

- SIPs - Shift-and-Invert Parallel Spectral Transformations
- SLEPc - scalable eigenvalue/eigenvector solver packages.
- TAO - scalable optimization algorithms
- veltisto ("optimum")- for problems with constraints which are time-independent pdes.

All have PETSc's style of programming

13

What Can We Handle?

- PETSc has run problem with 500 million unknowns
  http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf
- PETSc has run on over 6,000 processors efficiently
  ftp://info.mcs.anl.gov/pub/tech_reports/reports/P776.ps.Z
- PETSc applications have run at 2 Teraflops
  LANL PFLOTRAN code
- PETSc also runs on your laptop
- Only a handful of our users ever go over 64 processors

14

The PETSc Programming Model

- Distributed memory, “shared-nothing”
  - Requires only a standard compiler
  - Access to data on remote machines through MPI
- Hide within objects the details of the communication
- User orchestrates communication at a higher abstract level than direct MPI calls

PETSc Structure

Getting Started

```
PetscInitialize();
ObjCreate(MPI_comm,&obj);
ObjSetType(obj, );
ObjSetFromOptions(obj, );

ObjSolve(obj, );
ObjGetxxx(obj, );

ObjDestroy(obj);
PetscFinalize()
```

Integration

PETSc Numerical Components

Nonlinear Solvers (SNES)

Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers (TS)

Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods (KSP)

GMRES	CG	CGS	Bi-CG-STAB	TFQMR	Richardson	Chebychev	Other

Preconditioners (PC)

Additive Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

Matrices (Mat)

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

Distributed Arrays(DA)

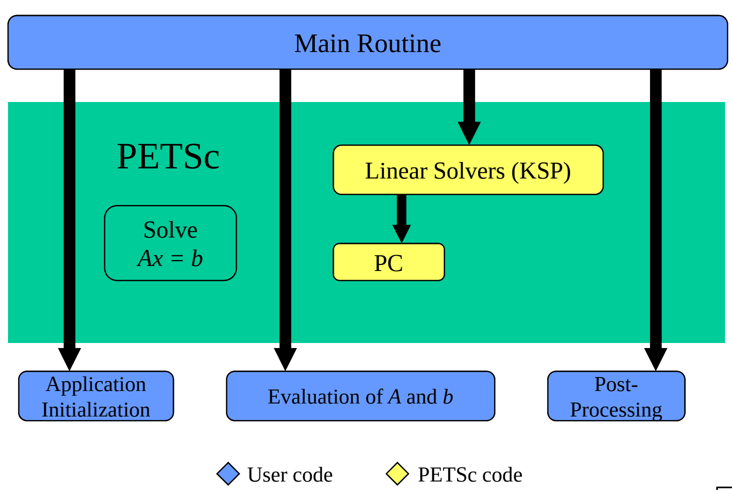
Index Sets (IS)

Indices	Block Indices	Stride	Other

Vectors (Vec)

17

Linear Solver Interface: KSP



beginner

solvers: linear

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

Recursion: Specifying Solvers for Schwarz Preconditioner Blocks

- Specify KSP 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

Summary of Proposed 3D Time Advance

$$B_j^0 U_j^{n+1} + D_j^0 + \varepsilon(A_j^1 U_{j+1}^{n+1} + B_j^1 U_j^{n+1} + C_j^1 U_{j-1}^{n+1} + D_j^1) = 0$$

U_j^{n+1} is vector of all unknown velocities on plane j at new time

$B_j^0, A_j^1, B_j^1, C_j^1$ are 2D sparse matrices at plane j

D_j^0, D_j^1 are 2D vectors at plane j

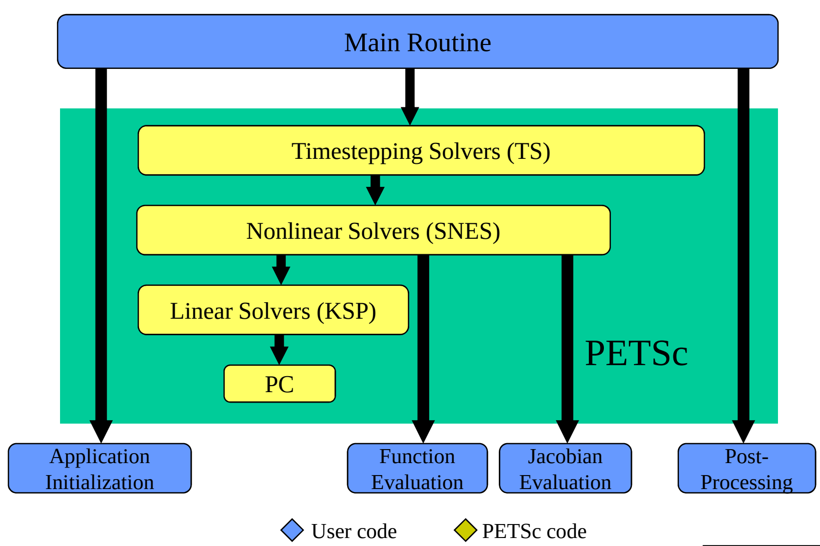
Possible iteration scheme. Use SuperLU to factor the B_j^0 simultaneously

$$U_j^{i+1} = -\left[B_j^0\right]^{-1} \cdot \left[D_j^0 + \varepsilon(A_j^1 U_{j+1}^i + B_j^1 U_j^i + C_j^1 U_{j-1}^i + D_j^1)\right]$$

Note that B_j^0 matrices only need to be factored once per timestep

21

Flow of Control for PDE Solution



PETSc Structure

Nonlinear Solver Interface: SNES

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)

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

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,…]

1

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

2

solvers: nonlinear

PETSc Programming Aids

- Correctness Debugging
  - Automatic generation of tracebacks
  - Detecting memory corruption and leaks
  - Optional user-defined error handlers
- Performance Profiling
  - Integrated profiling using -log_summary
  - Profiling by stages of an application
  - User-defined events

Integration

Ongoing Research and Developments

- Framework for unstructured meshes and functions defined over them
- Framework for multi-model algebraic system
- Bypassing the sparse matrix memory bandwidth bottleneck
  - Large number of processors (nproc =1k, 10k,…)
  - Peta-scale performance
- Parallel Fast Poisson Solver
- More TS methods
- …

27

Framework for Meshes and Functions Defined over Them

- The PETSc DA class is a topology and discretization interface.
  - Structured grid interface
    - Fixed simple topology
  - Supports stencils, communication, reordering
    - Limited idea of operators
- The PETSc Mesh class is a topology interface
  - Unstructured grid interface
    - Arbitrary topology and element shape
  - Supports partitioning, distribution, and global orders

28

- The PETSc DM class is a hierarchy interface.
  - Supports multigrid
    - DMMG combines it with the MG preconditioner
  - Abstracts the logic of multilevel methods

- The PETSc Section class is a function interface
  - Functions over unstructured grids
    - Arbitrary layout of degrees of freedom
  - Supports distribution and assembly

29

Creating a DA

DACreate2d(comm, wrap, type, M, N, m, n, dof, s, lm[], ln[], *da)

wrap: Specifies periodicity

DA_NONPERIODIC, DA_XPERIODIC, DA_YPERIODIC, …

type: Specifies stencil

DA_STENCIL_BOX, DA_STENCIL_STAR

M/N: Number of grid points in x/y-direction

m/n: Number of processes in x/y-direction

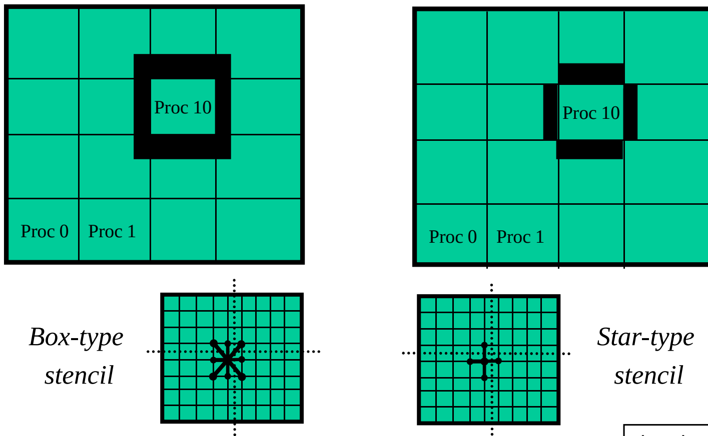
s: The stencil width

lm/ln: Alternative array of local sizes

30

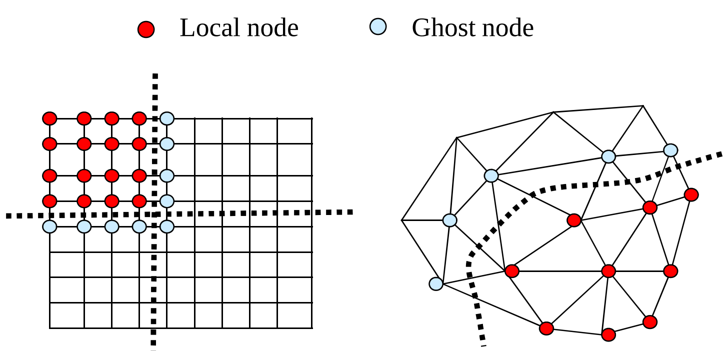
Distributed Arrays

Data layout and ghost values



data layout: distributed arrays

Ghost Values



To evaluate a local function *f(x)* , each process requires

- its local portion of the vector *x*
- its **ghost values** – bordering portions of *x* owned by neighboring processes.

data layout

A DA is more than a Mesh

A DA contains

topology, geometry, and an implicit Q1 discretization

It is used as a template to create

- Vectors (functions)
- Matrices (linear operator)

33

Creating the Mesh

- Generic object
  - MeshCreate()
  - MeshSetMesh()
- File input
  - MeshCreatePCICE()
  - MeshCreatePyLith()
- Generation
  - MeshGenerate()
  - MeshRefine()
  - ALE: :MeshBuilder::createSquareBoundary
- Representation
  - ALE::SieveBuilder::buildTopology()
  - ALE::SieveBuilder::buildCoordinates()
- Partitioning and distribution
  - MeshDistribute()
  - MeshDistributeByFace()

34

Parallel Sieves

- Sieves use names, not numberings
  - Numberings can be constructed on demand
- Overlaps relate names on different processes
  - An overlap can be encoded by a Sieve
- Distribution of a Section pushes forward along the Overlap
  - Sieves are distributed as “cone” sections

35

Sections associate data to submeshes

- Name comes from section of a fiber bundle
  - Generalizes linear algebra paradigm
- Define restrict(), update()
- Define complete()
- Assembly routines take a Sieve and several Sections
  - This is called a Bundle

36

Section Types

Section can contain arbitrary values

- C++ interface is templated over value type
- C interface has two value types
  - SectionReal
  - SectionInt

Section can have arbitrary layout

- C++ interface can place unknowns on any Mesh entity (Sieve point)
  - Mesh::setupField() parametrized by Discretization and BoundaryCondition
- C interface has default layouts
  - MeshGetVertexSectionReal()
  - MeshGetCellSectionReal()

37

Section Assembly

First we do local operations:

- Loop over cells
- Compute cell geometry
- Integrate each basis function to produce an element vector
- Call SectionUpdateAdd()

Then we do global operations:

- SectionComplete() exchanges data across overlap
  - C just adds nonlocal values (C++ is flexible)
- C++ also allows completion over arbitrary overlap

38

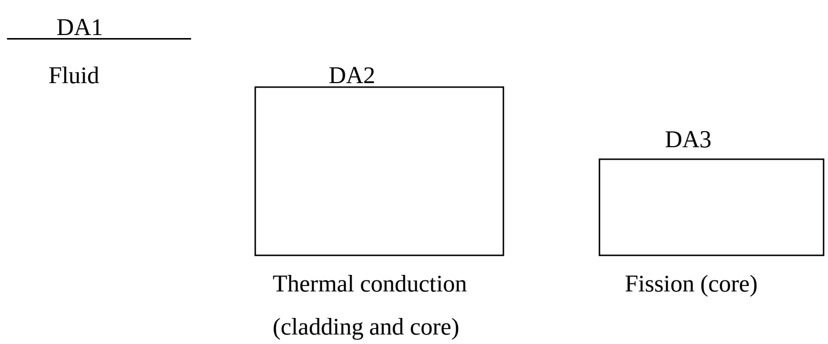
Framework for
Multi-model Algebraic System

~petsc/src/snes/examples/tutorials/ex31.c, ex32.c

Framework for Multi-model Algebraic System
~petsc/src/snes/examples/tutorials/ex31.c

A model "multi-physics" solver based on the Vincent Mousseau's reactor core pilot code:

There are three grids



```
/* Create the DMComposite object to manage the three grids/physics. */
DMCompositeCreate(app.comm,&app.pack);
DACreate1d(app.comm,DA_XPERIODIC,app.nxv,6,3,0,&da1);
DMCompositeAddDA(app.pack,da1);
DACreate2d(app.comm,DA_YPERIODIC,DA_STENCIL_STAR,…,&da2);
DMCompositeAddDA(app.pack,da2);
DACreate2d(app.comm,DA_XYPERIODIC,DA_STENCIL_STAR,…,&da3);
DMCompositeAddDA(app.pack,da3);

/* Create the solver object and attach the grid/physics info */
DMMGCreate(app.comm,1,0,&dmmg);
DMMGSetDM(dmmg,(DM)app.pack);
DMMGSetSNES(dmmg,FormFunction,0);

/* Solve the nonlinear system */
DMMGSolve(dmmg);

/* Free work space */
DMCompositeDestroy(app.pack);
DMMGDestroy(dmmg);
```

41

```
/* Unwraps the input vector and passes its local ghosted pieces into the user function */
FormFunction(SNES snes,Vec X,Vec F,void *ctx)
…
DMCompositeGetEntries(dm,&da1,&da2,&da3);
DAGetLocalInfo(da1,&info1);

/* Get local vectors to hold ghosted parts of X;
   then fill in the ghosted vectors from the unghosted global vector X */
DMCompositeGetLocalVectors(dm,&X1,&X2,&X3);
DMCompositeScatter(dm,X,X1,X2,X3);

/* Access subvectors in F - not ghosted and directly access the memory locations in F */
DMCompositeGetAccess(dm,F,&F1,&F2,&F3);

/* Evaluate local user provided function */
FormFunctionLocalFluid(&info1,x1,f1);
FormFunctionLocalThermal(&info2,x2,f2);
FormFunctionLocalFuel(&info3,x3,f3);
…
```

42

Bypassing the Sparse Matrix Memory Bandwidth Bottleneck

- Newton-multigrid provides
  - good nonlinear solver
  - easy utilization of software libraries
  - low computational efficiency
- Multigrid-Newton provides
  - good nonlinear solver
  - lower memory usage
  - potential for high computational efficiency
  - requires “code generation/in-lining”

43

- Parallel Fast Poisson Solver
- More TS methods
- …

44

How will we solve numerical applications in 20 years?

- Not with the algorithms we use today?
- Not with the software (development) we use today?

45

How Can We Help?

- Provide documentation:
  - http://www.mcs.anl.gov/petsc
- Quickly answer questions
- Help install
- Guide large scale flexible code development
- Answer email at petsc-maint@mcs.anl.gov

46