Portable, Extensible Toolkit for Scientific Computation (PETSc)

Hong Zhang

Computer Science, Illinois Institute of Technology

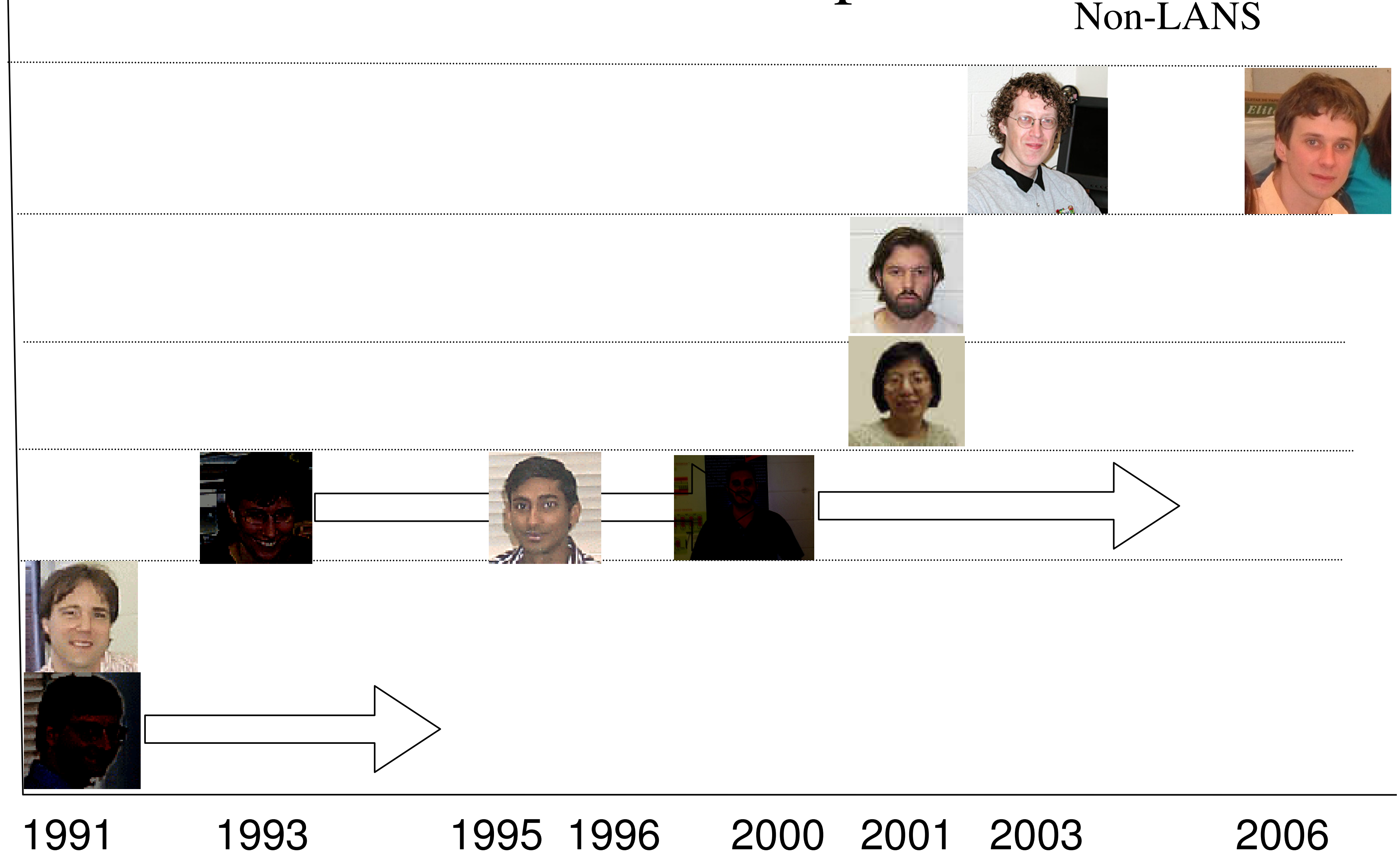
Mathematics and Computer Science, Argonne National Laboratory

April, 2007 at the Center for Computation & Technology, LSU

Outline

- Overview of PETSc
- Linear solver interface: KSP
- Nonlinear solver interface: SNES
- Profiling, tracing and viewing of computational objects
- Ongoing research and developments

Team and Active Developers



Original Goals of PETSc

- Provide software for the scalable (parallel) solution of algebraic systems arising from partial differential equation simulations.
  - Leverage inherited structure from the grid and the PDEs.
  - Eliminate the MPI from MPI programming!
  - Provide **wrappers** for other decent solver software.

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)

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

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.

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

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)

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)

Interfaced Packages

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

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

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

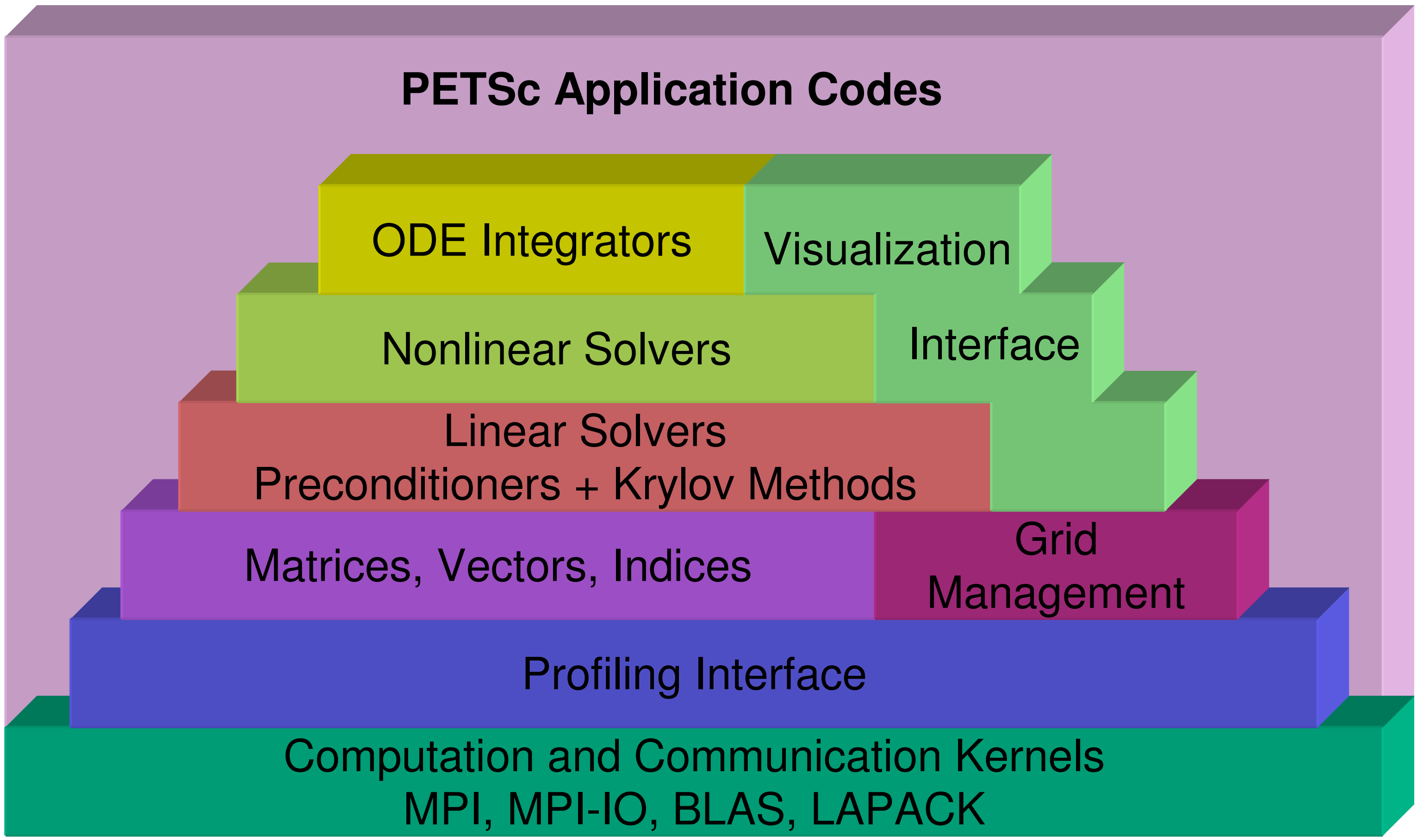
- PETSc applications have run at 2 Teraflops

  LANL PFLOTRAN code

- PETSc also runs on your laptop

- Only a handful of our users ever go over 64 processors

Structure of PETSc



PETSc Structure

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

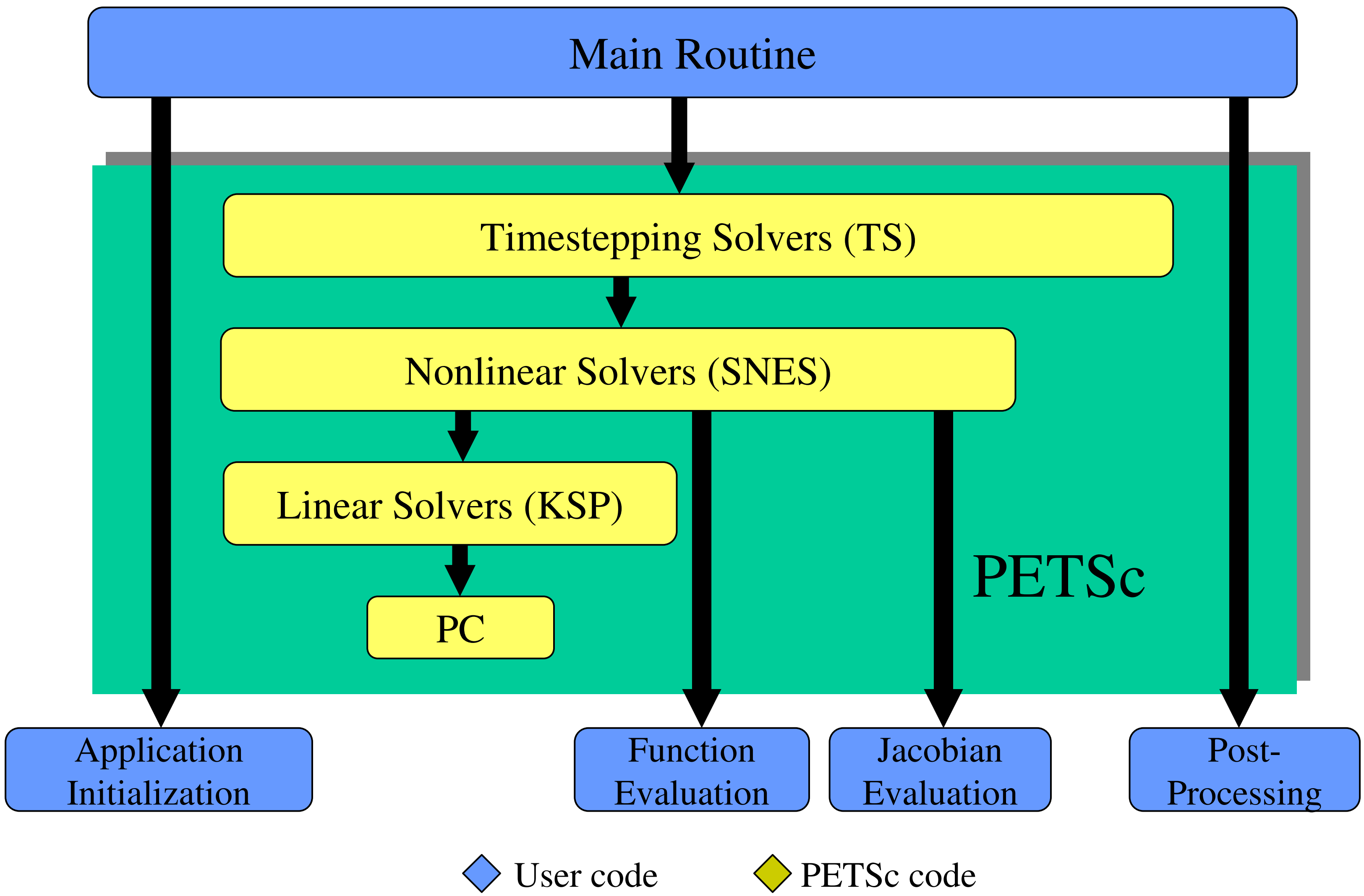
PETSc Structure

PETSc is only a Library

- PETSc is merely a set of library interfaces
  - You write main()
  - You control output
  - You control the basic flow of the program
  - We propagate the errors from underlying packages
  - We present (largely) the same interfaces in
    - C/C++
    - F77/F90

See Gropp in SIAM, OO Methods for Interop SciEng, '99

Flow of Control for PDE Solution



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)

Basic Linear Solver Code (C/C++)

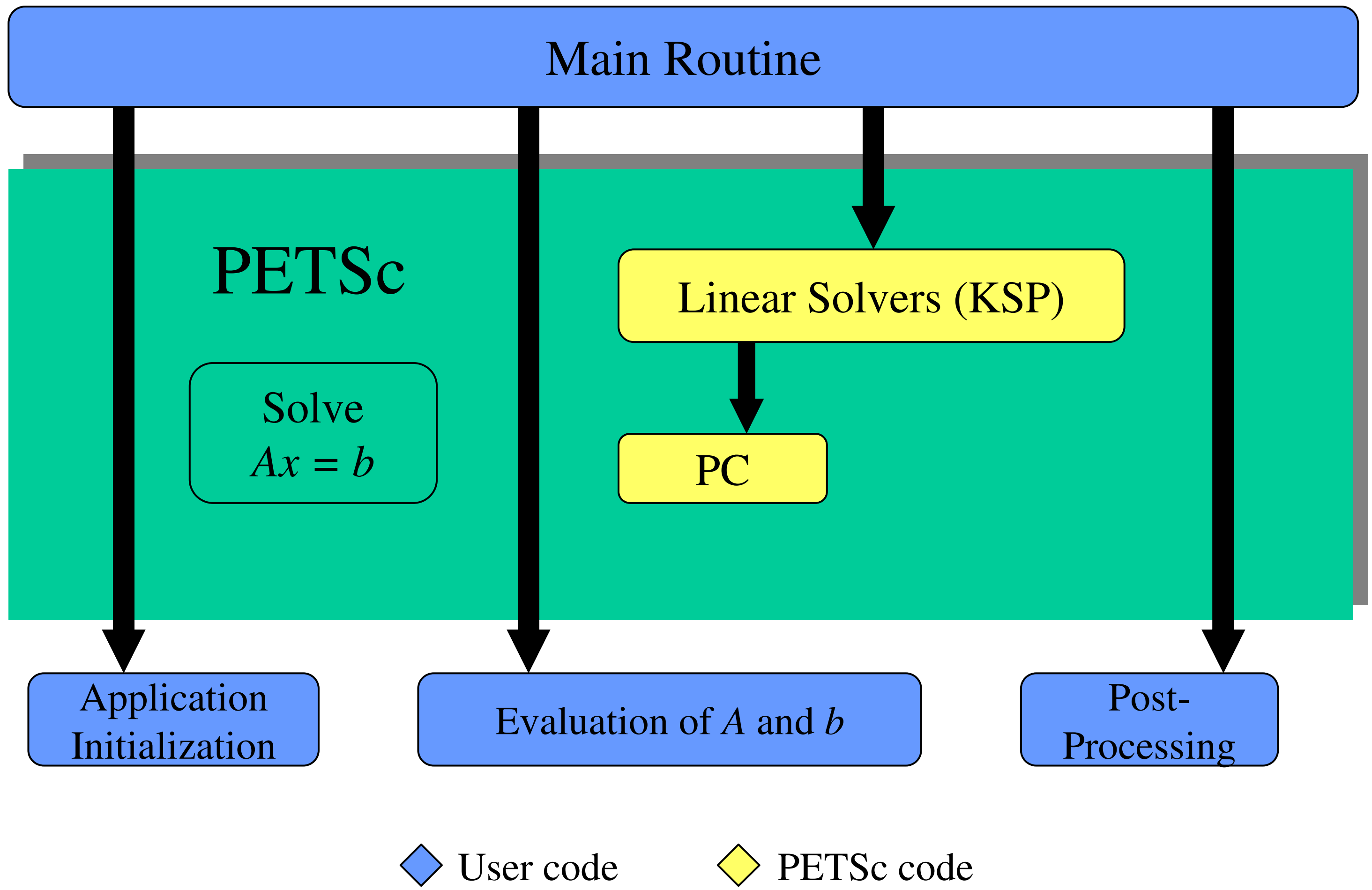
```
KSP    ksp;          /* linear solver context */
Mat    A;            /* matrix */
Vec    x, b;         /* solution, RHS vectors */
int    n, its;       /* problem dimension, number of iterations */

MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,n,n,&A);
MatSetFromOptions(A);
/* (code to assemble matrix not shown) */
VecCreate(PETSC_COMM_WORLD,&x);
VecSetSizes(x,PETSC_DECIDE, n);
VecSetFromOptions(x);
VecDuplicate(x,&b);
/* (code to assemble RHS vector not shown)*/

KSPCreate(PETSC_COMM_WORLD,&ksp);
KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
KSPSetFromOptions(ksp);
KSPSolve(ksp,b,x);
KSPDestroy(ksp);
```

Indicate whether the preconditioner has the same nonzero pattern as the matrix *each time a system is solved.* This default works with *all* preconditioners. Other values (e.g., SAME_NONZERO_PATTERN) can be used for particular preconditioners. Ignored when solving only one system

beginner

solvers: linear

Linear Solver Interface: KSP



beginner

solvers: linear

Example

~petsc/src/ksp/ksp/examples/tutorials/ex10.c

Linear Solvers in PETSc

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 (direct solve, sequential only)
- Arbitrary matrix
- etc.

beginner

solvers: linear

Customization Options

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

beginner

solvers: linear

Setting Solver Options at Runtime

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

1

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

2

1 beginner | 2 intermediate

solvers: linear

Linear Solvers: Monitoring Convergence

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

1

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

2

- User-defined monitors, using callbacks

3

1	2	3
beginner	intermediate	advanced

solvers:
linear

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

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

Debugging

Support for parallel debugging

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

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

Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- Active if PETSc was configured with

  --with-debugging=1 (default)
  - Can also profile application code segments
- Print summary data with option: -log_summary
- Print redundant information from PETSc routines: -info [infofile]
- Print the trace of the functions called: -log_trace [logfile]

profiling and
performance tuning

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

Finite Difference Jacobian Computation

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

Uniform access to all linear and nonlinear solvers

- -ksp_type [cg,gmres,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

Parallel Data Layout and Ghost Values

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

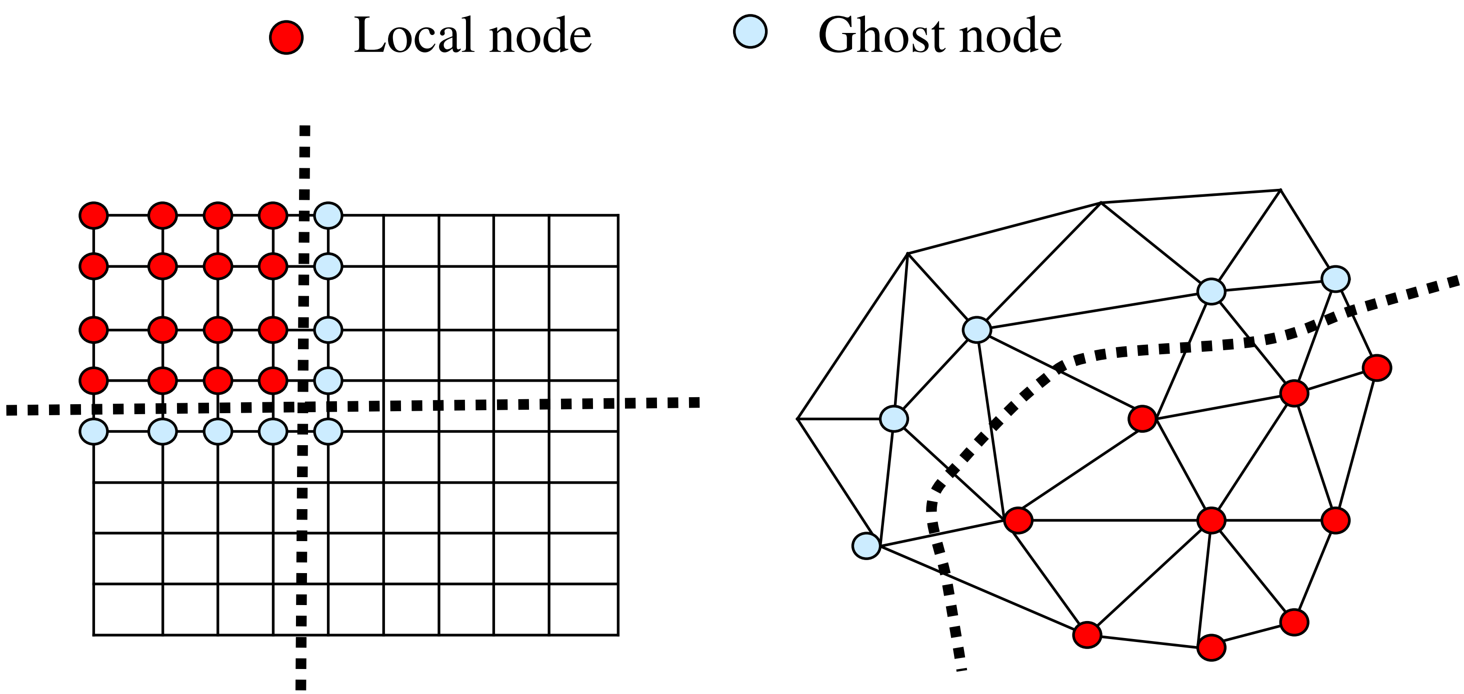
Usage Concepts

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

important concepts

data layout

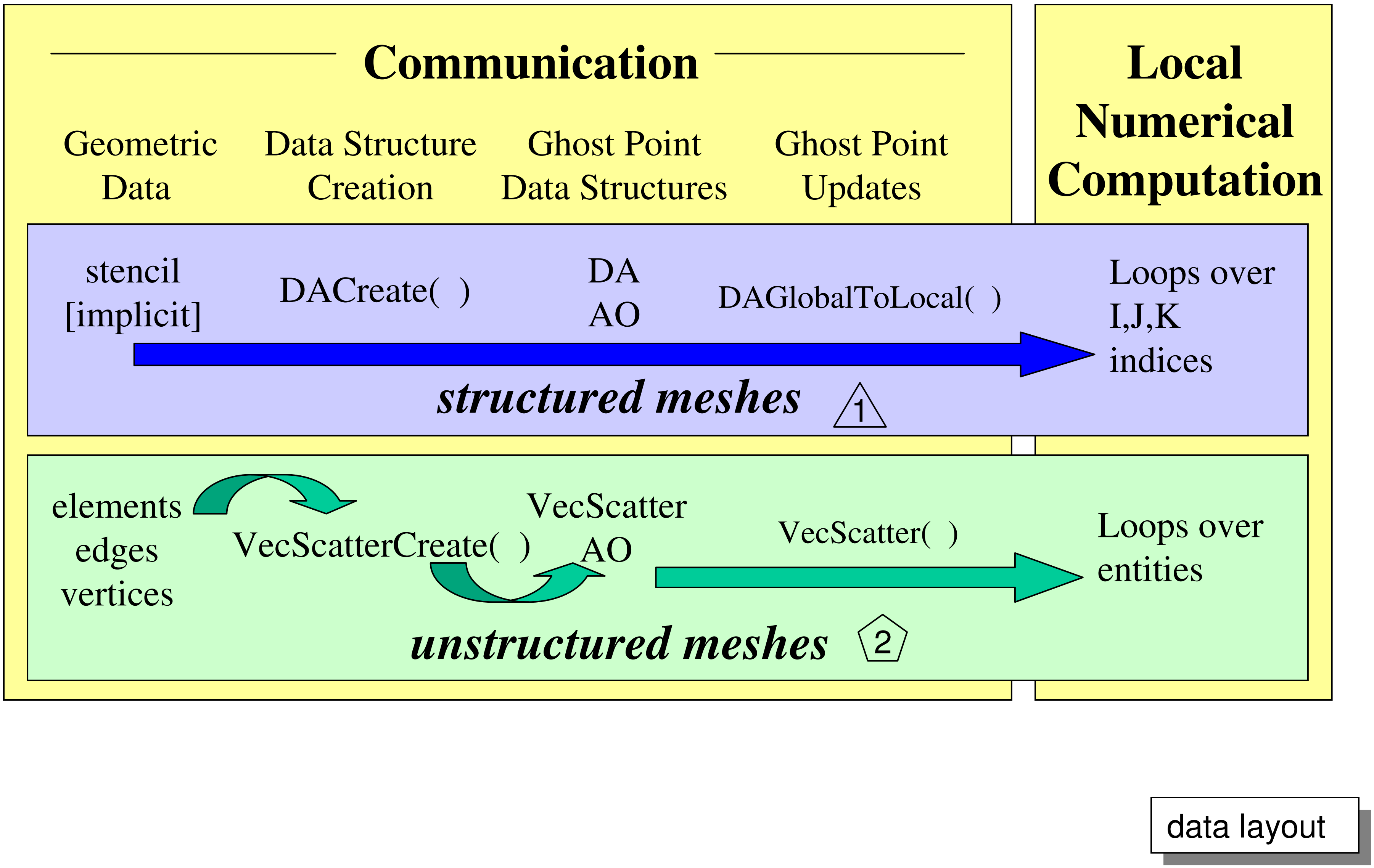
Ghost Values



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.

data layout

Communication and Physical Discretization

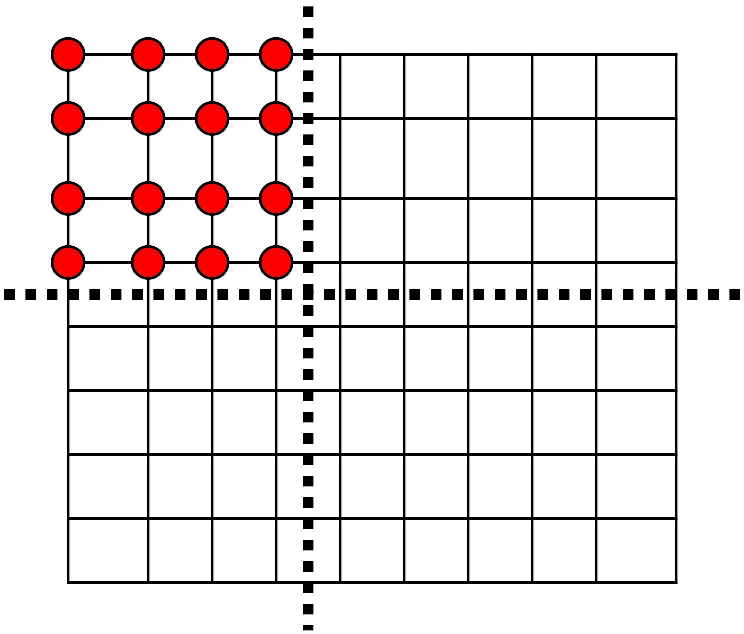


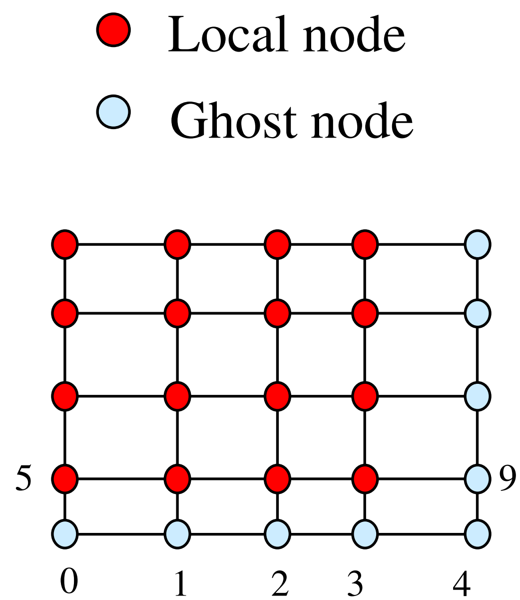
DA: Parallel Data Layout and Ghost Values for Structured Meshes

- Local and global indices
- Local and global vectors
- DA creation
- Ghost point updates
- Viewing

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

data layout:
distributed arrays

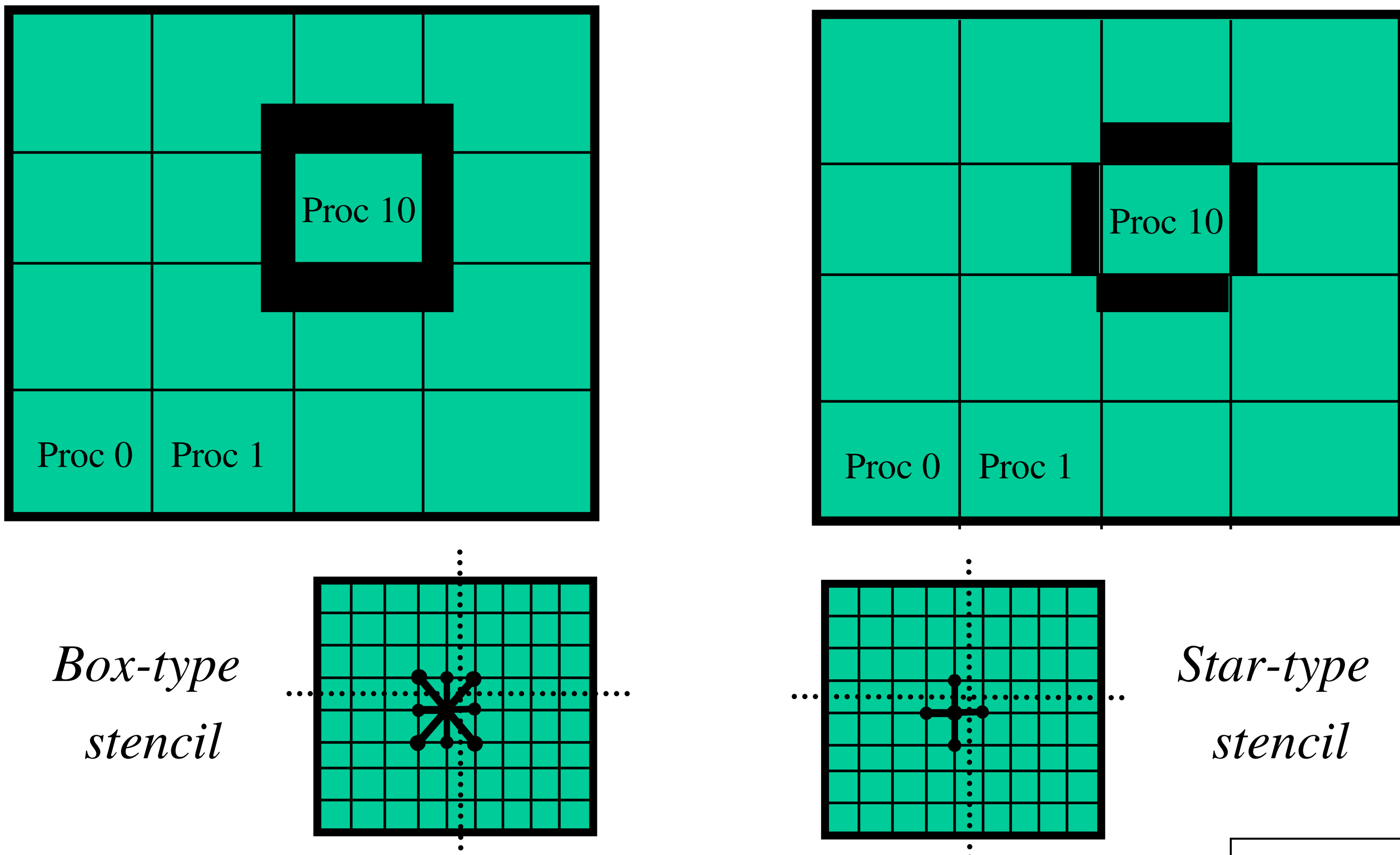
Logically Regular Meshes

- DA - Distributed Array: object containing information about vector layout across the processes and communication of ghost values
- Form a DA
  - DACreate1d(….,DA *)
  - DACreate2d(….,DA *)
  - DACreate3d(….,DA *)
- Create the corresponding PETSc vectors
  - DACreateGlobalVector(DA, Vec *) or
  - DACreateLocalVector(DA, Vec *)
- Update ghostpoints (scatter global vector into local parts, including ghost points)
  - DAGlobalToLocalBegin(DA, …)
  - DAGlobalToLocalEnd(DA,…)

data layout: distributed arrays

Distributed Arrays

Data layout and ghost values



data layout:
distributed arrays

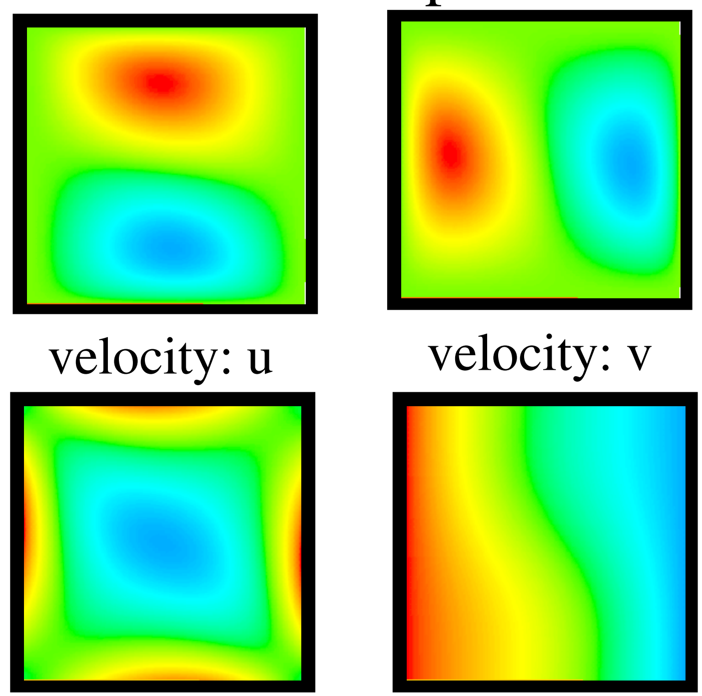
Sample Nonlinear Application:

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



velocity: u

velocity: v

vorticity: ζ

temperature: T

Application code author: D. E. Keyes

solvers: nonlinear

Ongoing Research and Developments

- Framework for multi-model algebraic system

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

- Framework for unstructured meshes and functions defined over them

- Bypassing the sparse matrix memory bandwidth bottleneck
  - Large number of processors (nproc =1k, 10k,…)
  - Peta-scale performance

- More TS methods

- …

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”

How will we solve numerical applications in 20 years?

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

References

- http://www.mcs.anl.gov/petsc