Introduction to PETSc 2

Matt Knepley

Kris Buschelman

Argonne National Laboratory

February 4, 2004

Tutorial Objectives

- Introduce the Portable, Extensible Toolkit for Scientific Computation (PETSc)
- Configure, Build, and Run a PETSc example
- Explain the different avenues available for extending PETSc functionality
- 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.

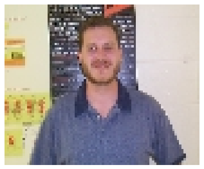
What is PETSc?

- 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

Building PETSc

Where is the Software?

- Tarball or Debian package
  - http://www.mcs.anl.gov/petsc
  - Also has patches
- Bitkeeper
  - bk://petsc.bkbits.net/petsc-dev
  - bk://sidl.bkbits.net/BuildSystem
- Accessories
  - Documentation tools
  - LAPACK and other package links

Configuration

- Set $PETSC_DIR to the installation root
  - Optionally set $PETSC_ARCH
- Running configure
  - `./config/configure.py`
  - `./config/darwin6.8.py`
  - `./config/configure.py --help`
  - `./config/configure.py --package-dirs=/usr/local`
- Configuring external packages
  - Each package has a separate module
  - `${PETSC_DIR}/python/PETSc/packages/MPI.py`

Extending Configuration

- See `python/PETSc/packages/Matlab.py`
- Add a module with config.base.Configure object
  - Should have `configure()` and `configureHelp()`
- Add code to check for package
- Add code to customize PETSc
  - `addSubstitution()`
  - `addDefine()`
- Add testing code at the end of the module

Building the Libraries

- Uses recursive make
  - `cd ${PETSC_DIR}; make BOPT=g`
- Check build when done
  - `cd ${PETSC_DIR}; make BOPT=g test`
- Can also build a subtree
  - `cd <subdir>; make BOPT=g ACTION=lib tree`
- Complete log for each build
  - `make_log_${PETSC_ARCH}_${BOPT}`
- Can build multiple sets of libraries
  - `${PETSC_DIR}/lib/lib$BOPT/$PETSC_ARCH/`

Connecting 3rd Party Packages

- Usually done by configure
- Information written to `bmake/${PETSC_ARCH}/packages`

```
PARMETIS_DIR         = /usr/local/ParMetis-3.0
PARMETIS_INCLUDE     = -I${PARMETIS_DIR}/include
PARMETIS_LIB         = -L ${PARMETIS_DIR}/lib -lparmetis -lmetis
PETSC_HAVE_PARMETIS = -DPETSC_HAVE_PARMETIS
```

- Requires additions to the build system internals
  - Add variables to `bmake/config/packages.in`
  - `bmake/common/variables`: edit `PCONF` and `EXTERNAL_LIB`

Automatic Downloads

- With 2.2.1, packages will be automatically
  - downloaded
  - configured and built
  - installed in PETSc
- Currently works for several packages
  - Sowing, lgrind, c2html
  - BLAS/LAPACK
  - MPICH
  - HYPRE
  - ParMetis
  - MUMPS (BLACS/ScalLAPACK)

Interfaced Solvers

1. LU (Sequential)
   1. SuperLU (Demmel and Li, LBNL)
   2. ESSL (IBM)
   3. Matlab
   4. LUSOL (from MINOS - Michael Saunders, Stanford)
2. Parallel LU
   1. SuperLU (Demmel and Li, LBNL)
   2. SPOOLES (Ashcroft, Boeing, was funded by ARPA)
   3. MUMPS (European)
3. Parallel Cholesky – DSCPACK (Raghavan, Penn. State)
4. XYTlib – parallel direct solver (Fischer and Tufo, ANL)
5. SPAI – Sparse approximate inverse (parallel)
   1. Parasails (Chow, part of Hypre, LLNL)
   2. SPAI 3.0 (the Grote/Barnard implementation)

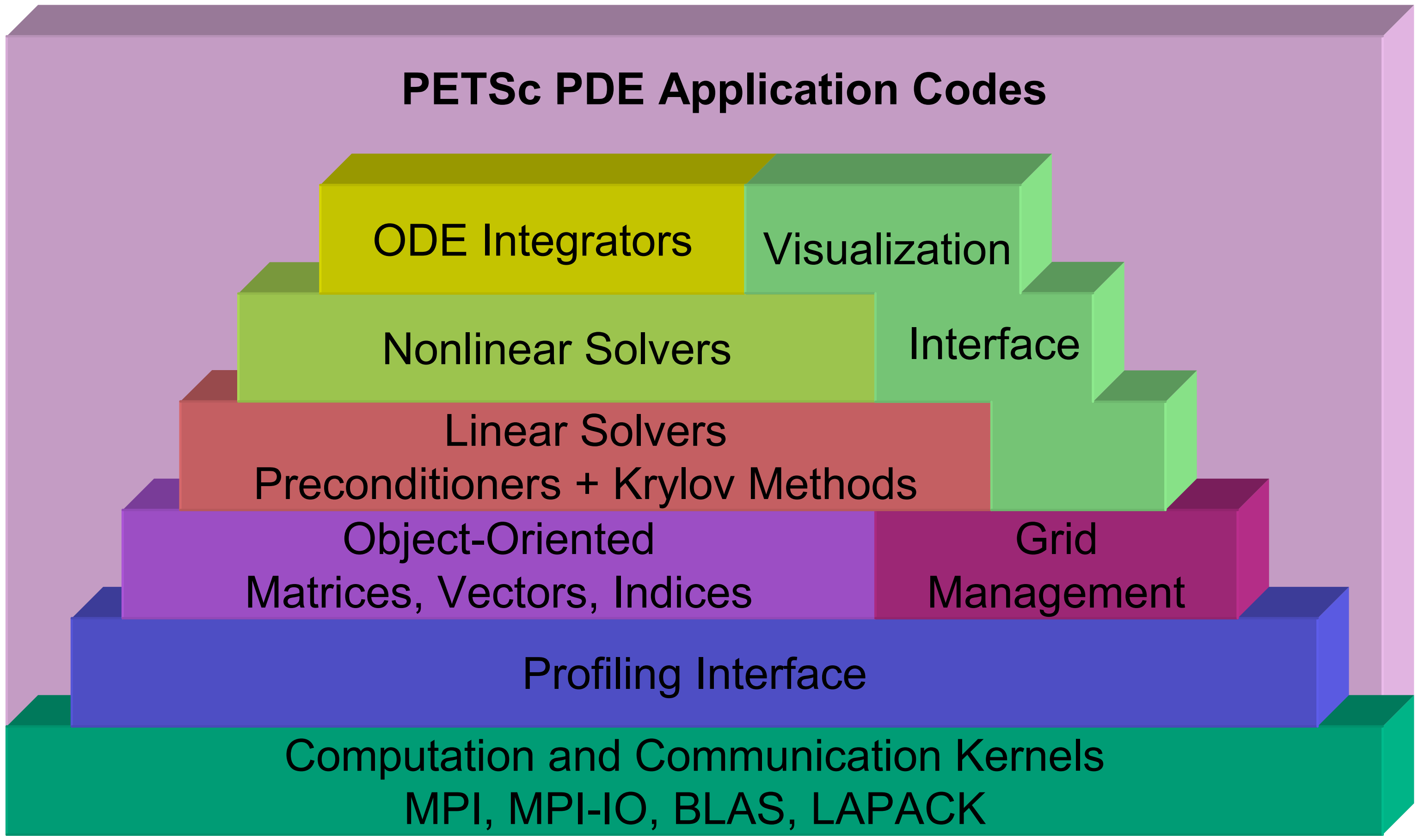
Building PETSc

Interfaced Solvers (continued)

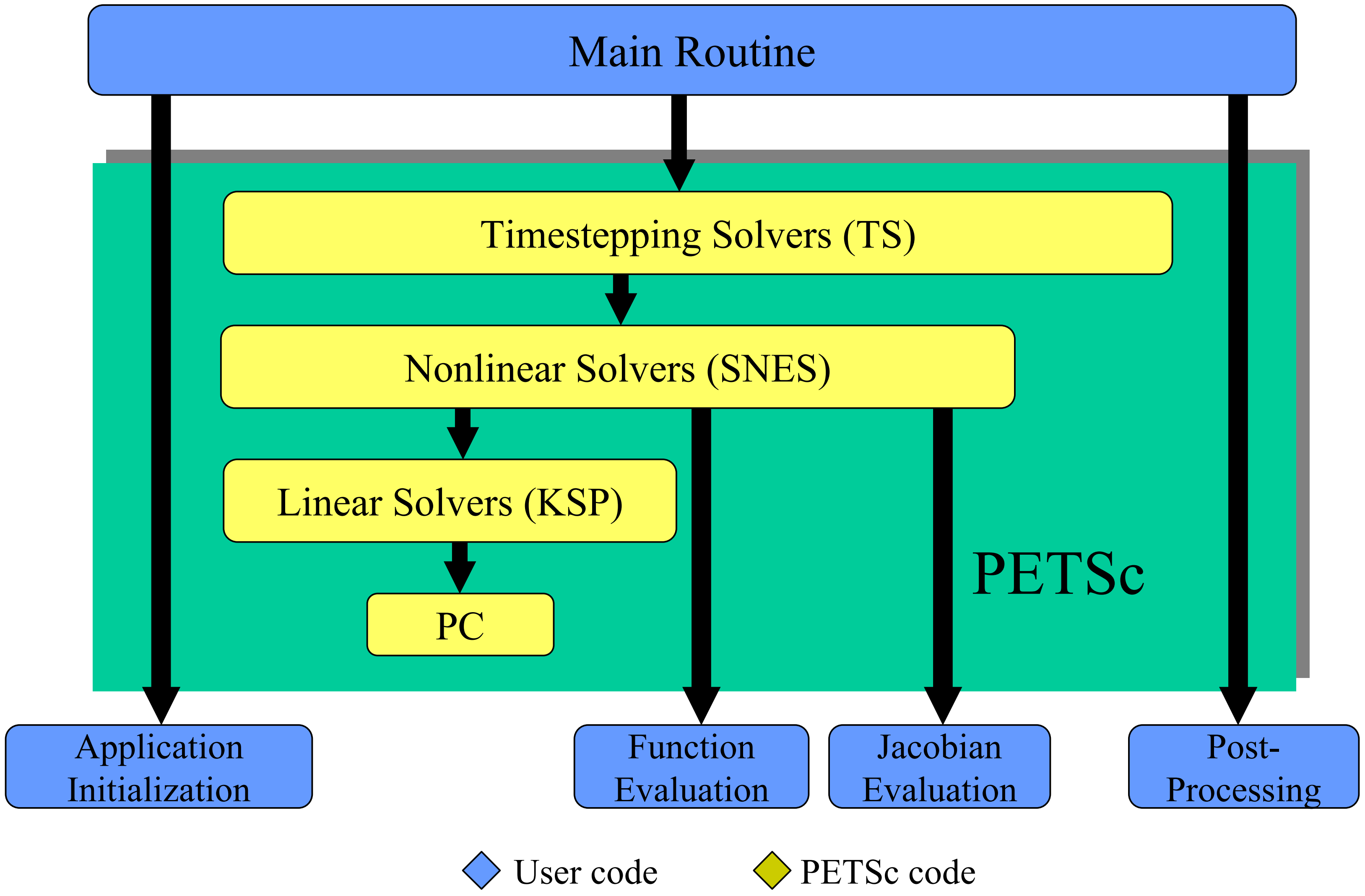
6. Algebraic multigrid
   1. Parallel BoomerAMG (part of Hypre, LLNL)
   2. Sequential RAMG (Ruge and Stueben's original code)
   3. Sequential SAMG (Stueben's modern version for systems of eqns)
7. Parallel ICC – BlockSolve95 (Jones and Plassman, ANL)
8. Parallel ILU
   1. BlockSolve95 (Jones and Plassman, ANL)
   2. PILUT (part of Hypre, LLNL)
   3. EUCLID (Hysom – also part of Hypre, ODU/LLNL)
9. Sequential ILUDT (part of Saad's SPARSEKIT2, U of MN)

The Structure of PETSc

Structure of PETSc


PETSc PDE Application Codes
ODE Integrators
Visualization
Interface
Nonlinear Solvers
Linear Solvers
Preconditioners + Krylov Methods
Object-Oriented
Matrices, Vectors, Indices
Grid
Management
Profiling Interface
Computation and Communication Kernels
MPI, MPI-IO, BLAS, LAPACK

Flow of Control for PDE Solution



PETSc Structure

Levels of Abstraction in Mathematical Software

- Application-specific interface
  - Programmer manipulates objects associated with the application
- High-level mathematics interface
  - Programmer manipulates mathematical objects
    - Weak forms, boundary conditions, meshes
- Algorithmic and discrete mathematics interface
  - Programmer manipulates mathematical objects
    - Sparse matrices, nonlinear equations
  - Programmer manipulates algorithmic objects
    - Solvers
- Low-level computational kernels
  - BLAS-type operations
  - FFT

PETSc emphasis

OO Programming & Design

Design based not on the data in object, but instead based on operations you perform with or on the data

For example a vector is not a 1d array of numbers but an abstract object where addition and scalar multiplication is defined

Added difficulty is the efficient use of the computer

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
  - Still exploits “compiler discovered” parallelism on each node
    - e.g., SMP
  - Hide within objects the details of the communication
  - User orchestrates communication at a higher abstract level

PETSc Structure

Collectivity

- MPI communicators (MPI_Comm) specify collectivity
  - Processes involved in a computation
- PETSc constructors are collective over a communicator
  - VecCreate(MPI_Comm comm, int m, int M, Vec *x)
  - Use **PETSC_COMM_WORLD** for all processes (like MPI_COMM_WORLD, but allows the same code to work when PETSc is started with a smaller set of processes)
- Some operations are collective, while others are not
  - collective: VecNorm()
  - not collective: VecGetLocalSize()
- If a sequence of collective routines is used, they **must** be called in the same order by each process.

Design Principles I

- Principle of Fairness
  - “If you can do it, your users will want to do it”
- Principle of Contrariness
  - “If you do it, your users will want to undo it”
- Both principles point to *symmetric* interfaces
  - Creation and query interfaces should be paired

Design Principles II

- The Hangover Principle
  - “You will not be smart enough to pick the solver”
  - “Never assume structure outside of the interface”
- Common in FE code
  - PETSc DA and HYPRE MatStruct?
  - We assume continuous fields that are *discretized*
  - It is unclear *what* structure a field must have
- Temptation to put metadata in a different place

Experimentation is Essential!

- Proof is not enough to examine solvers
  - N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, How fast are nonsymmetric matrix iterations?, SIAM J. Matrix Anal. Appl., 13:778--795, 1992.
  - Anne Greenbaum, Vlastimil Ptak, and Zdenek Strakos, Any Nonincreasing Convergence Curve is Possible for GMRES, SIAM J. Matrix Anal. Appl., 17 (3), pp.465-469, 1996.

What is not in PETSc?

- Higher level representations of PDEs
  - Unstructured mesh generation and manipulation
  - Discretizations
- Load balancing
- Sophisticated visualization capabilities
- Optimization and sensitivity

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

More is coming in PETSc 3!

Integrating PETSc

Application Interaction

- Be willing to experiment with algorithms
  - Optimality is rarely achieved without interplay between physics and algorithmics
- Adopt flexible, extensible programming
  - Algorithms and data structures not hardwired
- Be willing to play with the real code
- If possible, **profile** before seeking help
  - Automatic in PETSc

Integration

- PETSc is a set a library interfaces
  - We do not seize main()
  - We do not control output
  - We propagate errors from underlying packages
  - We present the same interfaces in:
    - C
    - C++
    - F77
    - F90

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

Initialization

- Call PetscInitialize()
  - Setup static data and services
  - Setup MPI if it is not already
- Call PetscFinalize()
  - Calculates logging summary
  - Shutdown and release resources
- Checks compile and link

Profiling

- -log_summary for a performance profile
  - Event timing
  - Memory usage
  - MPI messages
- Call PetscLogStagePush/Pop()
  - User can add new stages
- Call PetscLogEventBegin/End()
  - User can add new events

Command Line Processing

- Check for an option
  - PetscOptionsHasName()
- Retrieve a value
  - PetscOptionsGetInt(), PetscOptionsGetIntArray()
- Set a value
  - PetscOptionsSetValue()
- Clear, alias, reject, etc.

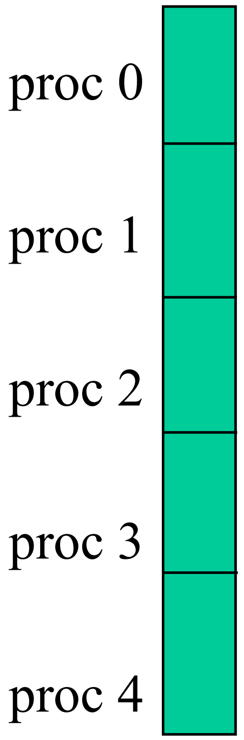
Linear Algebra I

- Vectors
  - Has a direct interface to the values
  - Supports all vector space operations
    - VecDot(), VecNorm(), VecScale()
  - Also unusual ops, e.g. VecSqrt(), VecInverse()
  - Automatic communication during assembly
  - Customizable communication (scatters)

Integration

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(MPI_Comm,Vec *)
    - MPI_Comm - processes that share the vector
  - VecSetSizes(Vec, int, int)
    - number of elements local to this process
    - or total number of elements
  - VecSetType(Vec,VecType)
    - Where VecType is
      - VEC_SEQ, VEC_MPI, or VEC_SHARED
    - VecSetFromOptions(Vec) lets you set the type at *runtime*



data objects: vectors

Creating a vector

Use PETSc to get value
from command line

```
Vec x;
int n;
…
PetscInitialize(&argc,&argv,(char*)0,help);
PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL);
…
VecCreate(PETSC_COMM_WORLD,&x);
VecSetSizes(x,PETSC_DECIDE,n);
VecSetType(x,VEC_MPI);
VecSetFromOptions(x);
```

Global size

PETSc determines
local size

data objects:
vectors

How Can We Use a PETSc Vector

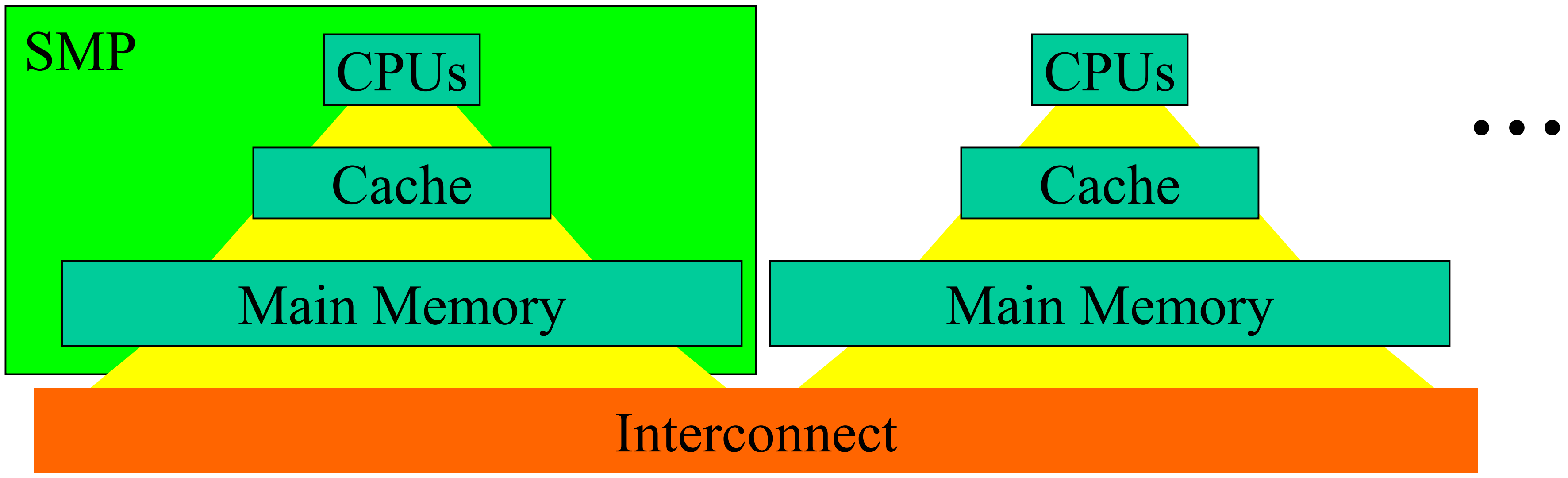
- PETSc supports "data structure-neutral" objects
  - distributed memory "shared nothing" model
  - single processors and shared memory systems
- PETSc vector is a "handle" to the real vector
  - Allows the vector to be distributed across many processes
  - To access the *elements* of the vector, we cannot simply do

    ```
    for (i=0; i<n; i++) v[i] = i;
    ```
  - We do not *require* that the programmer work only with the "local" part of the vector; we permit operations, such as setting an element of a vector, to be performed globally
- Recall how data is stored in the distributed memory programming model…

data objects: vectors

Sample Parallel System Architecture

- Systems have an increasingly deep memory hierarchy (1, 2, 3, and more levels of cache)
- Time to reference main memory 100's of cycles
- Access to shared data requires synchronization
  - Better to ensure data is local and unshared if possible

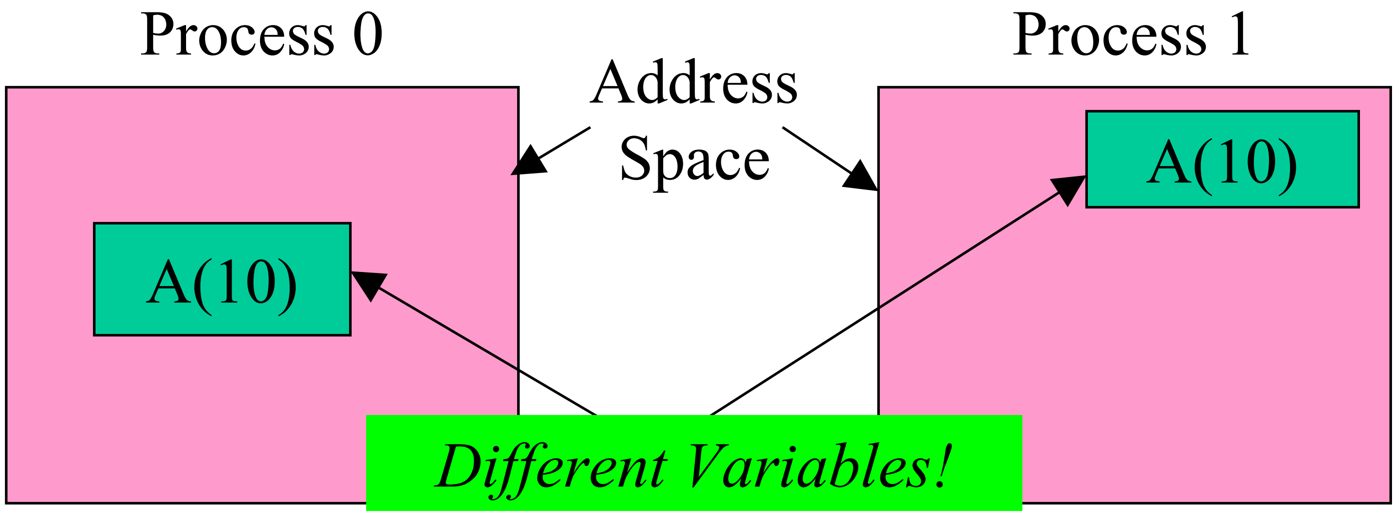


How are Variables in Parallel Programs Interpreted?

- Single process (address space) model
  - OpenMP and threads in general
  - Fortran 90/95 and compiler-discovered parallelism
  - System manages memory and (usually) thread scheduling
  - Named variables refer to the *same* storage
- Single name space model
  - HPF
  - Global Arrays
  - Data distribution part of the language, but programs still written as if there is a single name space
- Distributed memory (shared nothing)
  - Message passing
  - Names variables in different processes are *unrelated*

data objects:
vectors

Distributed Memory Model



- Integer A(10)

  …

  print *, A

- Integer A(10)
  do i=1,10
   A(i) = i
  enddo
  …

This A is completely different from this one

Vector Assembly

- A three step process
  - Each process tells PETSc what values to set or add to a vector component. Once *all* values provided,
  - Begin communication between processes to ensure that values end up where needed
  - (allow other operations, such as some computation, to proceed)
  - Complete the communication
- VecSetValues(Vec,…)
  - number of entries to insert/add
  - indices of entries
  - values to add
  - mode: [INSERT_VALUES,ADD_VALUES]
- VecAssemblyBegin(Vec)
- VecAssemblyEnd(Vec)

data objects:
vectors

Parallel Matrix and Vector Assembly

- Processes may generate any entries in vectors and matrices
- Entries need not be generated on the process on which they ultimately will be stored
- **PETSc automatically moves data during the assembly process if necessary**

One Way to Set the Elements of A Vector

-
  ```
  VecGetSize(x,&N);  /* Global size */
  MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
  if (rank == 0) {                        Vector index
     for (i=0; i<N; i++)
      VecSetValues(x,1,&i,&i,INSERT_VALUES);
  }                                        Vector value
  /* These two routines ensure that the data is
  distributed to the other processes */
  VecAssemblyBegin(x);
  VecAssemblyEnd(x);
  ```

data objects:
vectors

A Parallel Way to Set the Elements of A Distributed Vector

-
  ```
  VecGetOwnershipRange(x,&low,&high);
  for (i=low; i<high; i++)
       VecSetValues(x,1,&i,&i,INSERT_VALUES);
  /* These two routines must be called (in case some other process contributed a value owned by another process) */
  VecAssemblyBegin(x);
  VecAssemblyEnd(x);
  ```

data objects: vectors

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)	$r = \|\|x\|\|$

data objects:
vectors

A Complete PETSc Program

```
#include petscvec.h
int main(int argc,char **argv)
{
  Vec x;
  int n = 20,ierr;
  PetscTruth flg;
  PetscScalar one = 1.0, dot;

  PetscInitialize(&argc,&argv,0,0);
  PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL);
  VecCreate(PETSC_COMM_WORLD,&x);
  VecSetSizes(x,PETSC_DECIDE,n);
  VecSetFromOptions(x);
  VecSet(&one,x);
  VecDot(x,x,&dot);
  PetscPrintf(PETSC_COMM_WORLD,"Vector length %dn",(int)dot);
  VecDestroy(x);
  PetscFinalize();
  return 0;
}
```

data objects:
vectors

Working With Local Vectors

- It is sometimes more efficient to directly access the storage for the local part of a PETSc Vec.
  - E.g., for finite difference computations involving elements of the vector
- PETSc allows you to access the local storage with
  - `VecGetArray(Vec, double *[])`
- You must return the array to PETSc when you finish
  - `VecRestoreArray(Vec, double *[])`
- Allows PETSc to handle data structure conversions
  - For most common uses, these routines are inexpensive and do *not* involve a copy of the vector.

Example of VecGetArray

```
Vec      vec;
Double *avec;
…
VecCreate(PETSC_COMM_SELF,&vec);
VecSetSizes(vec,PETSC_DECIDE,n);
VecSetFromOptions(vec);
VecGetArray(vec,&avec);
/* compute with avec directly, e.g., */
PetscPrintf(PETSC_COMM_WORLD,
 "First element of local array of vec in each process is
%f\n", avec[0] );
VecRestoreArray(vec,&avec);
```

data objects:
vectors

Linear Algebra II

- Matrices
  - Must use MatSetValues()
    - Automatic communication
  - Supports many data types
    - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
  - Supports structures for many packages
    - Spooles, MUMPS, SuperLU, UMFPack, DSCPack

Integration

Matrices

- What are PETSc matrices?
  - Fundamental objects for storing linear operators (e.g., Jacobians)
- Create matrices via
  - MatCreate(…,Mat *)
    - MPI_Comm - processes 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.
    - MatSetFromOptions(Mat) lets you set the MatType at *runtime*.

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.
- A matrix is defined by its *interface*, the operations that you can perform with it.
  - Not by its data structure

data objects: matrices

Matrix Assembly

- Same form as for PETSc Vectors:
- 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)

data objects: matrices

Matrix Assembly Example

simple 3-point stencil for 1D discretization

```
Mat      A;
int      column[3], i;
double value[3];
…
MatCreate(PETSC_COMM_WORLD,
             PETSC_DECIDE,PETSC_DECIDE,n,n,&A);

MatSetFromOptions(A);
/* mesh interior */
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
if (rank == 0) {  /* Only one process creates matrix */
   for (i=1; i<n-2; 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  (code for global row 0 and n-1 omitted) */
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

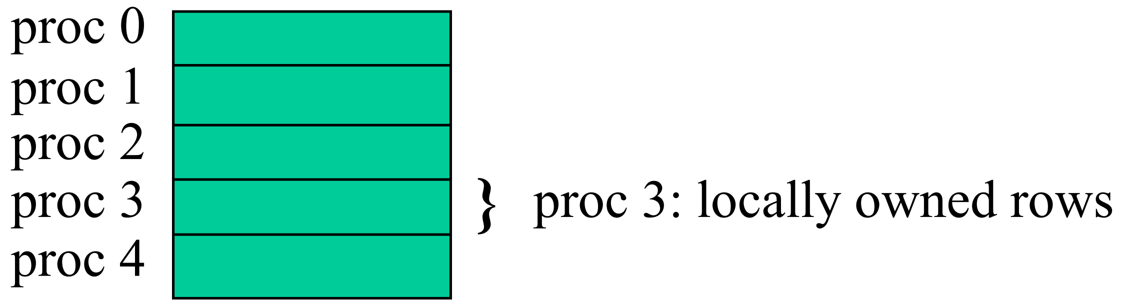
Choose the global
Size of the matrix

Let PETSc decide how
to allocate matrix
across processes

data objects:
matrices

Parallel Matrix Distribution

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



MatGetOwnershipRange(Mat A, int *rstart, int *rend)

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

data objects: matrices

Matrix Assembly Example With Parallel Assembly

simple 3-point stencil for 1D discretization

```
Mat     A;
int     column[3], i, start, end,istart,iend;
double value[3];
…
MatCreate(PETSC_COMM_WORLD,
          PETSC_DECIDE,PETSC_DECIDE,n,n,&A);
MatSetFromOptions(A);
MatGetOwnershipRange(A,&start,&end);
/* mesh interior */
istart = start; if (start == 0) istart = 1;
iend = end; if (iend == n-1) iend = n-2;
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
for (i=istart; i<iend; 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  (code for global row 0 and n-1 omitted) */
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

Choose the global Size of the matrix

Let PETSc decide how to allocate matrix across processes

data objects: matrices

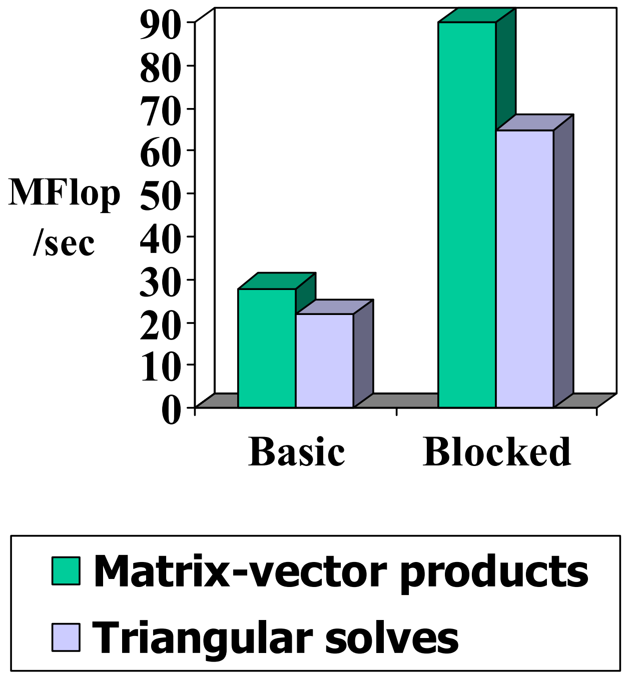
Why Are PETSc Matrices The Way They Are?

- No one data structure is appropriate for all problems
  - Blocked and diagonal formats provide significant performance benefits
  - PETSc provides a large selection of formats and makes it (relatively) easy to extend PETSc by adding new data structures
- Matrix assembly is difficult enough without being forced to worry about data partitioning
  - PETSc provides parallel assembly routines
  - Achieving high performance still requires making most operations local to a process but programs can be incrementally developed.
- Matrix decomposition by consecutive rows across processes is simple and makes it easier to work with other codes.
  - For applications with other ordering needs, PETSc provides "Application Orderings" (AO), described later.

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

data objects: matrices

Solver Categories

- **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

Integration

Linear Solvers

- Krylov Methods
  - Using PETSc linear algebra, just add:
    - KSPSetOperators(), KSPSetRhs(), KSPSetSolution()
    - KSPSolve()
  - Preconditioners must obey PETSc interface
    - Basically just the KSP interface
  - Can change solver dynamically from the cmd line

Nonlinear Solvers

- Using PETSc linear algebra, just add:
  - SNESSetFunction(), SNESSetJacobian()
  - SNESSolve()
- Can access subobjects
  - SNESGetKSP()
  - KSPGetPC()
- Can customize subobjects from the cmd line
  - Could give –sub_pc_type ilu, which would set the subdomain preconditioner to ILU

Higher Level Abstractions

- DA
  - Structured grid interface
    - Fixed simple topology
  - Supports stencils, communication, reordering
    - No idea of operators
  - Nice for simple finite differences

Integration

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

Integration

Debugging

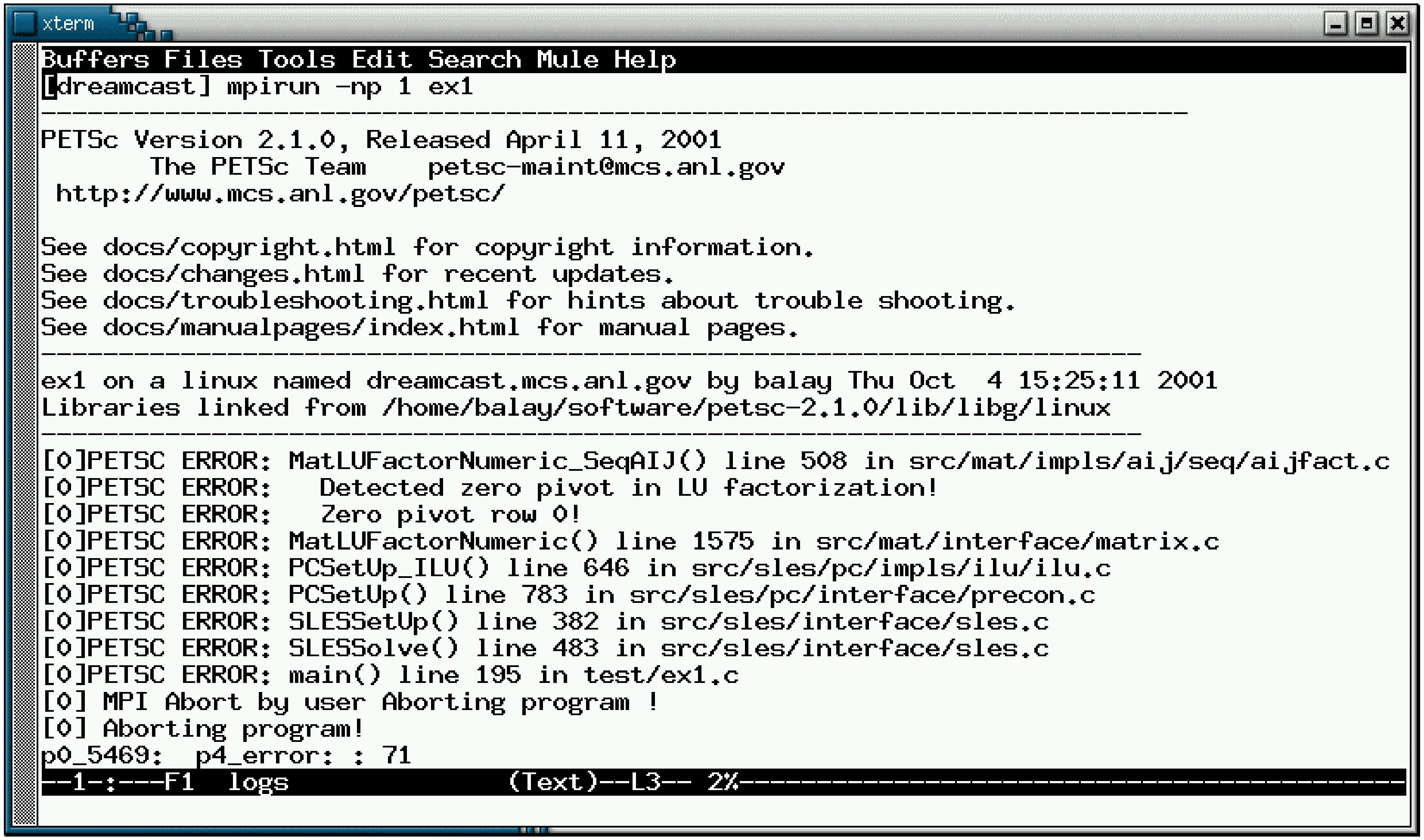
Support for parallel debugging

- -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( ).

Sample Error Traceback

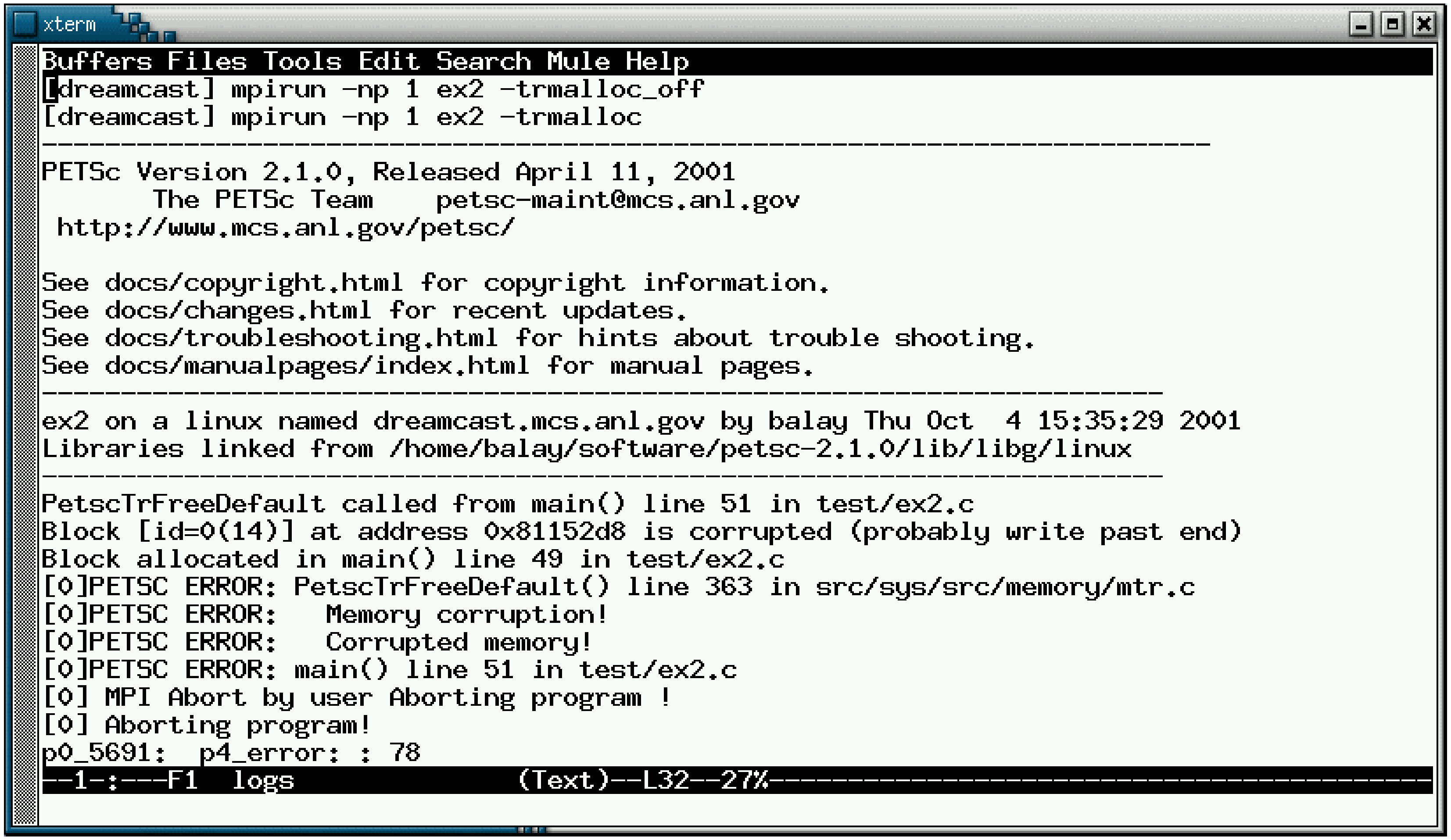
Breakdown in ILU factorization due to a zero pivot



```
[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%------------------------------------------
```

debugging and errors

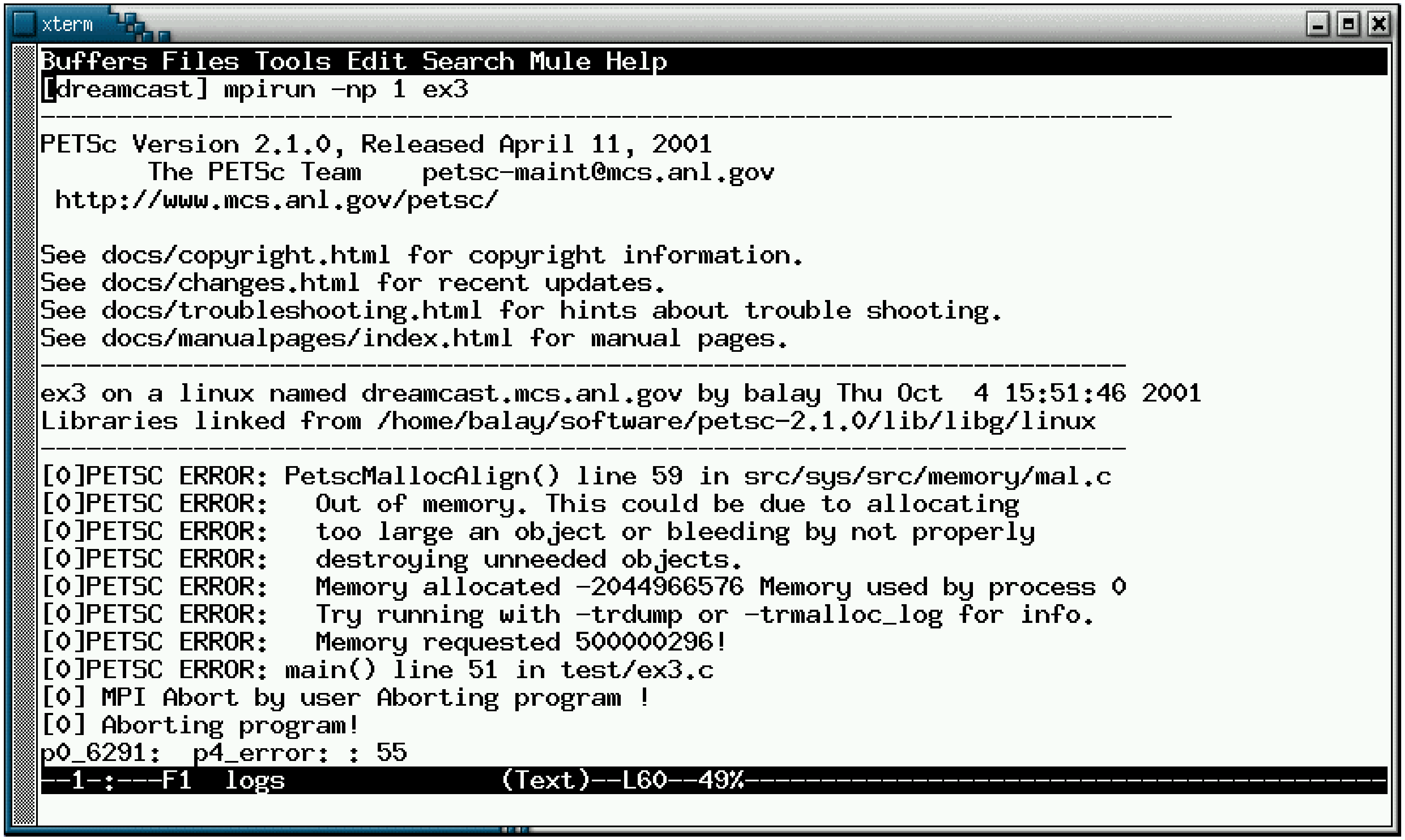
Sample Memory Corruption Error



```
[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%------------------------------------
```

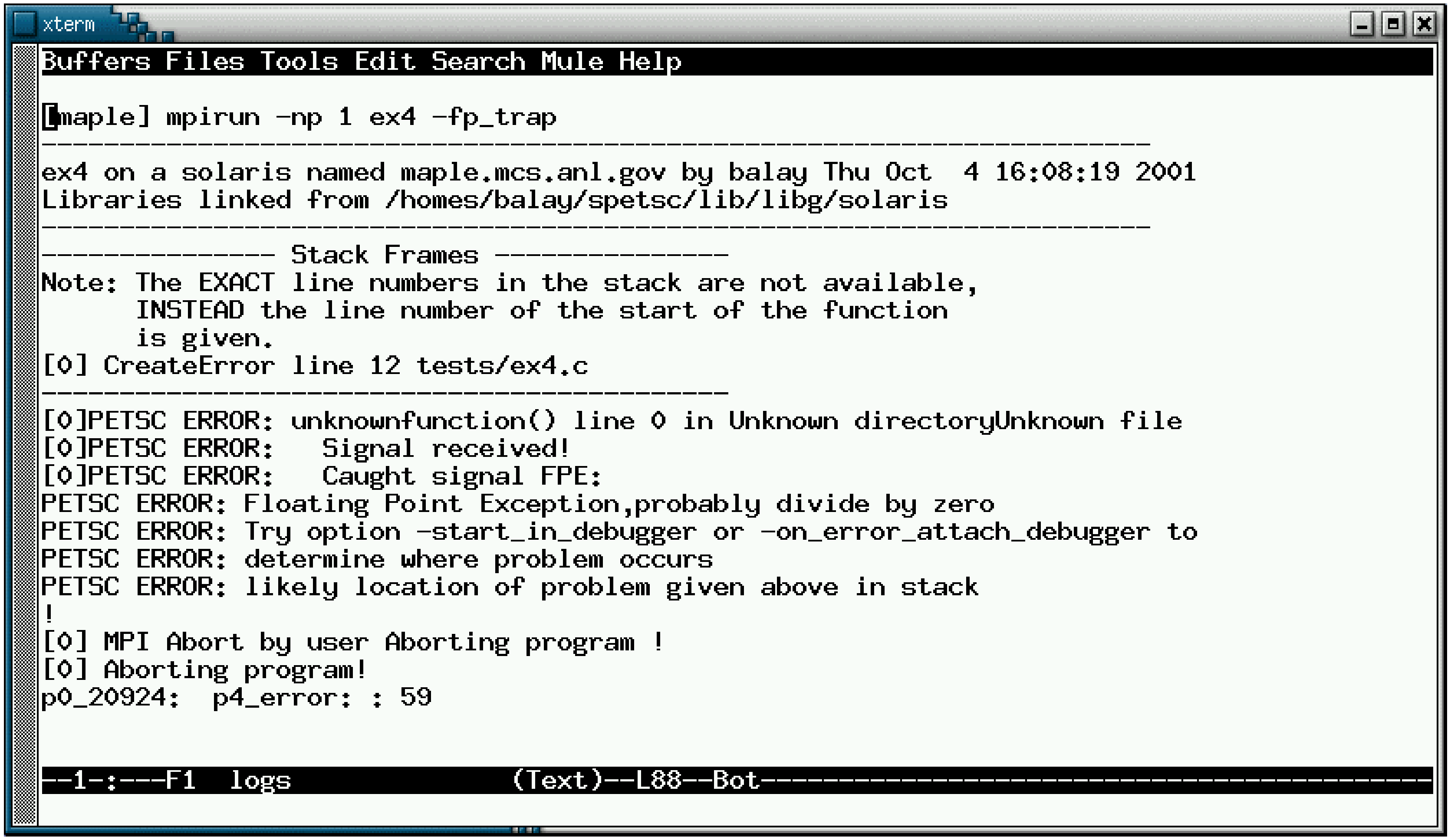
debugging and errors

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%------------------------------------------
```

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----------------------------------------
```

Profiling and Performance Tuning

Profiling:

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

Performance Tuning:

- Matrix optimizations
- Application optimizations
- Algorithmic tuning

profiling and performance tuning

Profiling

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

profiling and performance tuning

Sample -log_summary

```
------------------------------------------------------------------------------------------------------------------------
Event                Count      Time (sec)     Flops/sec                         --- Global ---  --- Stage ---   Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------

--- Event Stage 0: Main Stage

PetscBarrier           2 1.0 1.1733e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0

--- Event Stage 1: SetUp

VecSet                 2 1.0 9.3448e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatMultTranspose       1 1.0 1.8022e-03 1.0 1.85e+08 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0 57  0  0  0    185
MatAssemblyBegin       3 1.0 1.0057e-05 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0      0
MatAssemblyEnd         3 1.0 2.0356e-02 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   5  0  0  0  0      0
MatFDColorCreate       2 1.0 1.5341e-01 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 4.6e+01  1  0  0  0 16  36  0  0  0 74      0

--- Event Stage 2: Solve

VecDot                 2 1.0 3.2985e-03 1.0 9.56e+07 1.0 0.0e+00 0.0e+00 2.0e+00  0  0  0  0  1   0  0  0  0  2     96
VecMDot               45 1.0 9.3093e-02 1.0 1.59e+08 1.0 0.0e+00 0.0e+00 1.5e+01  0  0  0  0  5   1  1  0  0 19    159
VecNorm              112 1.0 2.0851e-01 1.0 8.47e+07 1.0 0.0e+00 0.0e+00 5.2e+01  1  1  0  0 18   2  1  0  0 64     85
```

```
MatMultTranspose   1 1.0 1.8022e-03 1.0 1.85e+08 ...
VecNorm          112 1.0 2.0851e-01 1.0 8.47e+07 ... 5.2e+01 …
```

More –log_summary

```
Memory usage is given in bytes:

Object Type          Creations   Destructions   Memory  Descendants' Mem.

--- Event Stage 0: Main Stage


--- Event Stage 1: SetUp

   Distributed array     4              0           0     2.37475e+06
           Index Set   104             24     2376480     0
                 Map    40             10        2000     0
                 Vec    36             10     2846384     0
         Vec Scatter    12              0           0     0
IS Local to global mapping      8              0           0      0
              Matrix     8              0           0     0
  Matrix FD Coloring     4              0           0     0
                SNES     4              0           0     0
       Krylov Solver    10              0           0     0
      Preconditioner    10              0           0     0

--- Event Stage 2: Solve

   Distributed array     0              4      822496     3.16488e+06
           Index Set    20            100     3578544     0
                 Map    26             56       11200     0
                 Vec   160            186    92490656     2864
         Vec Scatter     0             12     2374784     0
```

Still more –log_summary

```
=================================================================================
Average time to get PetscTime(): 1.13389e-08
Compiled without FORTRAN kernels
Compiled with double precision matrices (default)
sizeof(short) 2 sizeof(int) 4 sizeof(long) 4 sizeof(void*) 4
Libraries compiled on Fri May 28 01:39:58 PDT 2004 on MBuschel
Machine characteristics: CYGWIN_NT-5.1 MBuschel 1.5.9(0.112/4/2) 2004-03-18 23:05
Using PETSc directory: /home/Kris/petsc/petsc-dev
Using PETSc arch: cygwin
-----------------------------------------
Using C compiler: gcc -Wall -O -fomit-frame-pointer -Wno-strict-aliasing -I/home/K
c-dev/bmake/cygwin -I/home/Kris/petsc/petsc-dev/include   -I/software/MPI/mpich-nt

EXTERN_CXX  -D__SDIR__='. '
C Compiler version:
gcc (GCC) 3.3.1 (cygming special)\nCopyright (C) 2003 Free Software Foundation,
```

Adding a new Stage

```
int stageNum;
PetscLogStageRegister(&stageNum,"name");


PetscLogStagePush(stageNum);
[code to monitor]
PetscLogStagePop();
```

Adding a new Event

```
static int USER_EVENT;
PetscLogEventRegister(&USER_EVENT,”name”,CLASS_COOKIE);


PetscLogEventBegin(USER_EVENT,0,0,0,0);
[code to monitor]
PetscLogFlops(user_evnet_flops);
PetscLogEventEnd(USER_EVENT,0,0,0,0);
```

Adding a new Class

- Cookie identifies a class uniquely

```
static int CLASS_COOKIE;
PetscLogClassRegister(&CLASS_COOKIE,”name”);
```

- This initialization must happen before any objects of this type are created

Performance Requires Managing Memory

- Real systems have many levels of memory
  - Programming models try to hide memory hierarchy
    - Except C—`register`
- Simplest model: Two levels of memory
  - Divide at largest (relative) latency gap
  - Processes have their own memory
    - Managing a processes memory is known (if unsolved) problem
  - Exactly matches the distributed memory model

Sparse Matrix-Vector Product

- Common operation for optimal (in floating-point operations) solution of linear systems
- Sample code:

```
for row=0,n-1
    m   = i[row+1] - i[row];
    sum = 0;
    for k=0,m-1
        sum += *a++ * x[*j++];
    y[row] = sum;
```

- Data structures are a[nnz], j[nnz], i[n], x[n], y[n]

Simple Performance Analysis

- Memory motion:
  - nnz (sizeof(double) + sizeof(int)) + n (2*sizeof(double) + sizeof(int))
  - Perfect cache (never load same data twice)
- Computation
  - nnz multiply-add (MA)
- Roughly 12 bytes per MA
- Typical WS node can move ½-4 bytes/MA
  - *Maximum* performance is 4-33% of peak

More Performance Analysis

- Instruction Counts:
  - nnz (2*load-double + load-int + mult-add) +
    n (load-int + store-double)
- Roughly 4 instructions per MA
- Maximum performance is 25% of peak (33% if MA overlaps one load/store)
- Changing matrix data structure (e.g., exploit small block structure) allows reuse of data in register, eliminating some loads (x and j)
- Implementation improvements (tricks) cannot improve on these limits

profiling and
performance tuning

Alternative Building Blocks

- Performance of sparse matrix - multi-vector multiply:

Format	***Number of Vectors***	***Mflops***	
		Ideal	***Achieved***
AIJ	1	49	45
AIJ	4	182	120
BAIJ	1	64	55
BAIJ	4	236	175

- Results from 250 MHz R10000 (500 MF/sec peak)
- BAIJ is a block AIJ with blocksize of 4
- Multiple right-hand sides can be solved in nearly the same time as a single RHS

Matrix Memory Pre-allocation

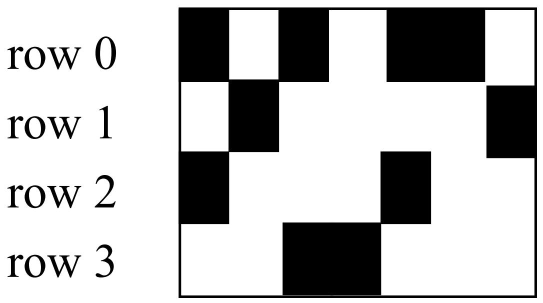
- PETSc sparse matrices are dynamic data structures. Can add additional nonzeros freely
- Dynamically adding many nonzeros
  - requires additional memory allocations
  - requires copies
  - can kill performance
- Memory pre-allocation provides the freedom of dynamic data structures plus good performance

Indicating Expected Nonzeros

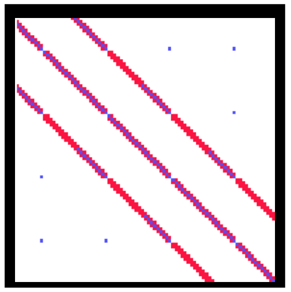
Sequential Sparse Matrices

MatCreateSeqAIJ(…., int *nnz,Mat *A)

- nnz[0] - expected number of nonzeros in row 0
- nnz[1] - expected number of nonzeros in row 1



sample nonzero pattern



another sample nonzero pattern

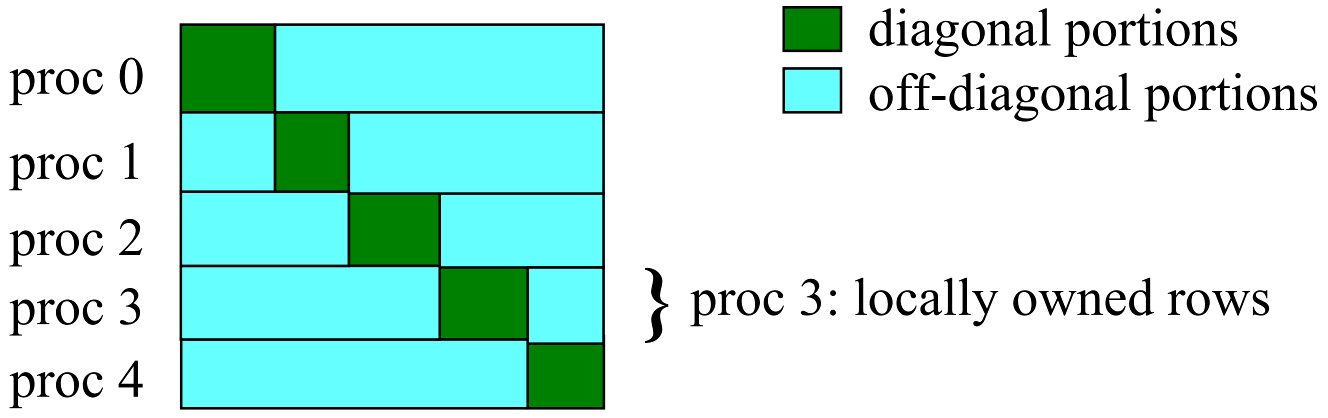
profiling and performance tuning

Symbolic Computation of Matrix Nonzero Structure

- Create matrix with MatCreate()
- Set type with MatSetType()
- Form the nonzero structure of the matrix
  - loop over the grid for finite differences
  - loop over the elements for finite elements
  - etc.
- Preallocate matrix
  - MatSeqAIJSetPreallocation()
  - MatMPIAIJSetPreallocation()

Parallel Sparse Matrices

- Each process locally owns a submatrix of contiguously numbered global rows.
- Each submatrix consists of **diagonal** and **off-diagonal** parts.



profiling and performance tuning

Indicating Expected Nonzeros

Parallel Sparse Matrices

```
MatMPIAIJSetPreallocation(Mat A,
                          int d_nz, int *d_nnz,
                          int o_nz, int *o_nnz)
```

- `d_nnz[]` - expected number of nonzeros per row in diagonal portion of local submatrix
- `o_nnz[]` - expected number of nonzeros per row in off-diagonal portion of local submatrix

Verifying Predictions

- Use runtime option: -log_info
- Output:

  [proc #] Matrix size: %d X %d; storage space: %d unneeded, %d used

  [proc #] Number of mallocs during MatSetValues( ) is %d

```
[merlin] mpirun ex2 -log_info
[0]MatAssemblyEnd_SeqAIJ:Matrix size: 56 X 56; storage space:
[0]    310 unneeded, 250 used
[0]MatAssemblyEnd_SeqAIJ:Number of mallocs during MatSetValues() is 0
[0]MatAssemblyEnd_SeqAIJ:Most nonzeros in any row is 5
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routines
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routines
Norm of error 0.000156044 iterations 6
[0]PetscFinalize:PETSc successfully ended!
```

Extending PETSc

Layering over PETSc

- TAO and Veltisto
- Use PETSc object structure
  - Dynamic dispatch
- Use dynamic linking facilities
  - Runtime type selection
- Use debugging tools
  - Memory management, runtime type checking

Linking to PETSc

- Application program or framework
- Nothing but the libraries
  - Custom link by user
- Using the PETSc build variables
  - Include bmake/common/variables
- Using the PETSc build rules
  - Include bmake/common/base
  - Also makes available 3rd party packages

Customizing the Framework

- Adding additional logging
  - Pushing new stages
  - Registering new events and logging classes
- Adding 3rd party PETSc libraries
  - Enlarging the dynamic link path

Adding an Implementation

- See `src/ksp/pc/impls/jacobi/jacobi.c`
- Implement the interface methods
  - For Jacobi, PCSetUp(), PCApply(),…
- Define a constructor
  - Allocate and initialize the class structure
  - Fill in the function table
  - Must have C linkage
- Register the constructor
  - See `src/ksp/ksp/interface/dlregis.c`
  - Maps a string (class name) to the constructor
  - Usually uses PetscFListAdd()

Adding a Wrapper

- See `src/ts/impls/implicit/pvode/petscpvode.c`
- Just like an Implementation
  - Methods dispatch to 3rd party software
- Need to alter local makefile
  - Add a `requirespackage` line
  - Add include variable to `CPPFLAGS`
- Also need usual configure build additions

Adding a Subtype

- See `src/mat/impls/aij/seq/umfpack/umfpack.c`
- Have to virtualize methods by hand
- Define a constructor
  - Change type name first to correct name
  - Call MatSetType() for base type
  - Replace (and save) overidden methods
  - Construct any specific data
- Must also define a conversion to the base type
  - Only called in destructor right now

Adding a Type I

- See `src/ksp/ksp/kspimpl.h`
- Define a methods structure (interface)
  - A list of function pointers
- Define a type structure
  - First member is `PETSCHEADER(struct _Ops)`
  - Possibly other data members common to the type
  - A `void *data` for implementation structures

Adding a Type II

- See `src/ksp/ksp/interface/dlregis.c`
- Define a package initializer (`PetscDLLibraryRegister`)
  - Called when the DLL is loaded
    - Also called from generic create if linking statically
  - Registers classes and events (see below)
  - Registers any default implementation constructors
  - Setup any info or summary exclusions

Adding a Type III

- See `src/ksp/ksp/interface/itcreate.c`
- Define a generic create
  - Call package initializer if linking statically
  - Call `PetscHeaderCreate()`
  - Initialize any default data members
- Define a `setType()` method
  - Call the destructor of any current implementation
  - Call the constructor of the given implementation
  - Set the type name

Adding a Type IV

- Things swept under the rug
  - Want a `setFromOptions()` which allows selection of implementations at runtime
  - Have to manage the database of registered constructors
  - View and destroy functions handled a little funny due to historical reasons

Nonlinear Solvers (SNES)

SNES: Scalable Nonlinear Equations Solvers

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

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)

solvers: nonlinear

Nonlinear Solvers (SNES)

- Newton-based methods, including
  - Line search strategies
  - Trust region approaches (using TAO)
  - Pseudo-transient continuation
  - Matrix-free variants
- Can customize all phases of solution process

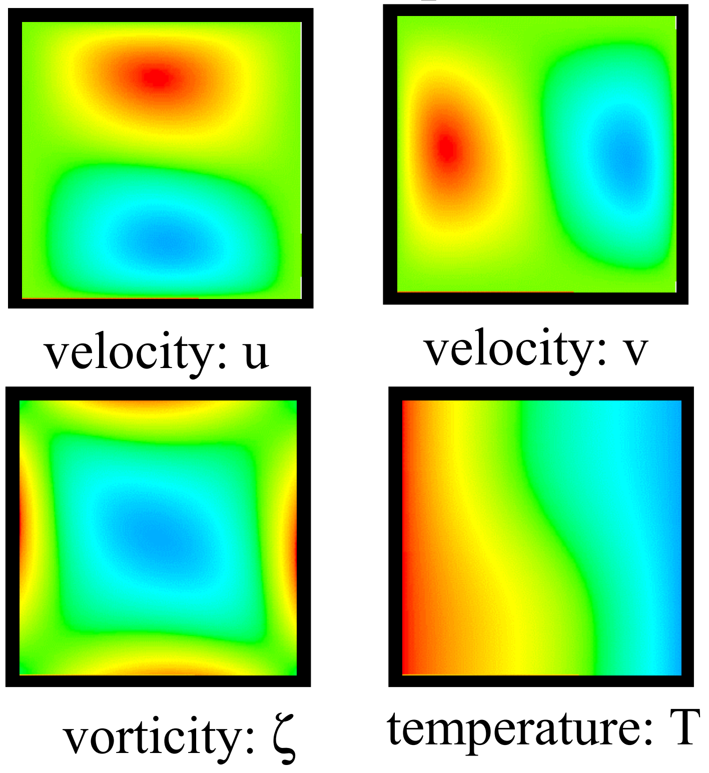
Sample Nonlinear Application:

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

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(PETSC_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,n,n,&J);
MatSetFromOptions(J);
VecCreate(PETSC_COMM_WORLD,&x);
VecSetSizes(x,PETSC_DECIDE,n);
VecSetFromOptions(x);
VecDuplicate(x,&F);

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

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

solvers:
nonlinear

Sample Application Context:

Driven Cavity Problem

```
typedef struct {
   /* - - - - - - - - - - - - - -  basic application data  - - - - - - - - - - - - - - - - - */
   double        lid_velocity, prandtl;    /* problem parameters */
   double        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;
```

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 */
    ….
    /* (Code to communicate nonlocal ghost point data not shown) */
    VecGetArray( F, &f );
    /* (Code to compute local function components; insert into f[ ] shown on
    next slide) */
    VecRestoreArray( F, &f );
    ….
return 0;
}
```

solvers:
nonlinear

Sample Local Computational Loops:

Driven Cavity Problem

```
#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;
 } }
 ….
```

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

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

Customization via Callbacks:
Setting a user-defined line search routine

SNESSetLineSearch(SNES snes,int(*ls)(…),void *lsctx)

Available line search routines ls( ) include:

– SNESCubicLineSearch( ) - cubic line search
– SNESQuadraticLineSearch( ) - quadratic line search
– SNESNoLineSearch( ) - full Newton step
– SNESNoLineSearchNoNorms( ) - full Newton step but calculates no norms (faster in parallel, useful when using a fixed number of Newton iterations instead of usual convergence testing)
– YourOwnFavoriteLineSearchRoutine( )

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

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, …
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> …

And many more options...

solvers:
nonlinear

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

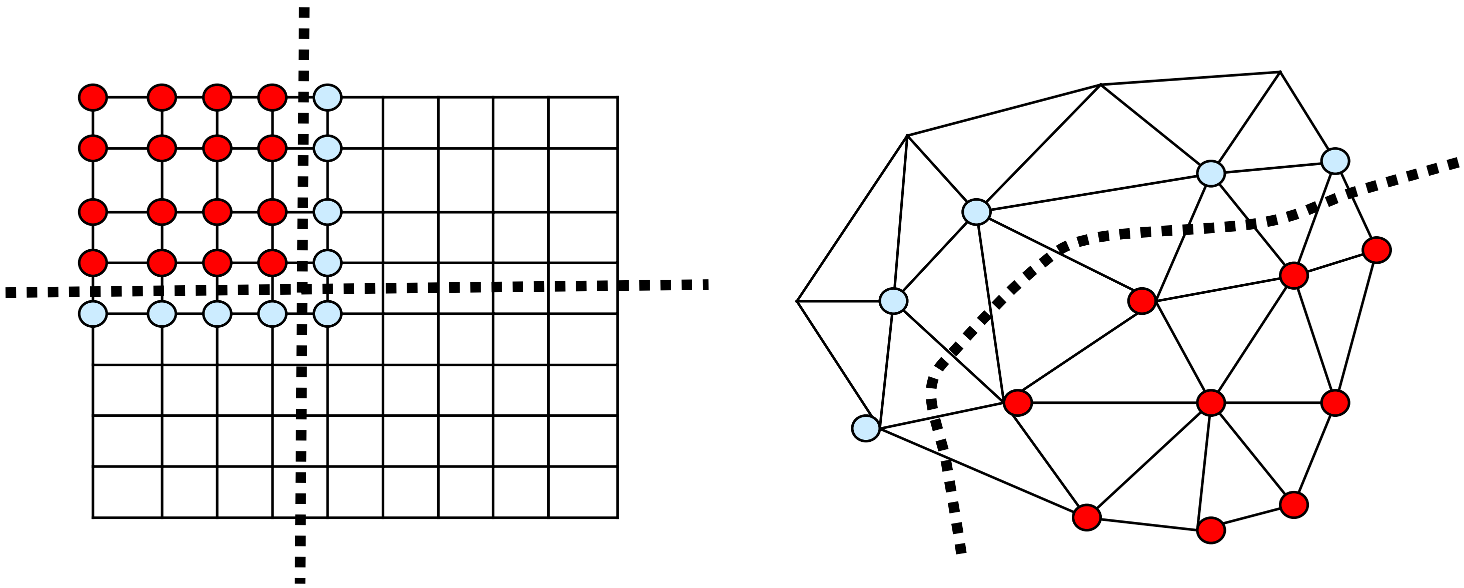
- Geometric data
- Data structure creation
- Ghost point updates
- Local numerical computation

important concepts

data layout

Ghost Values

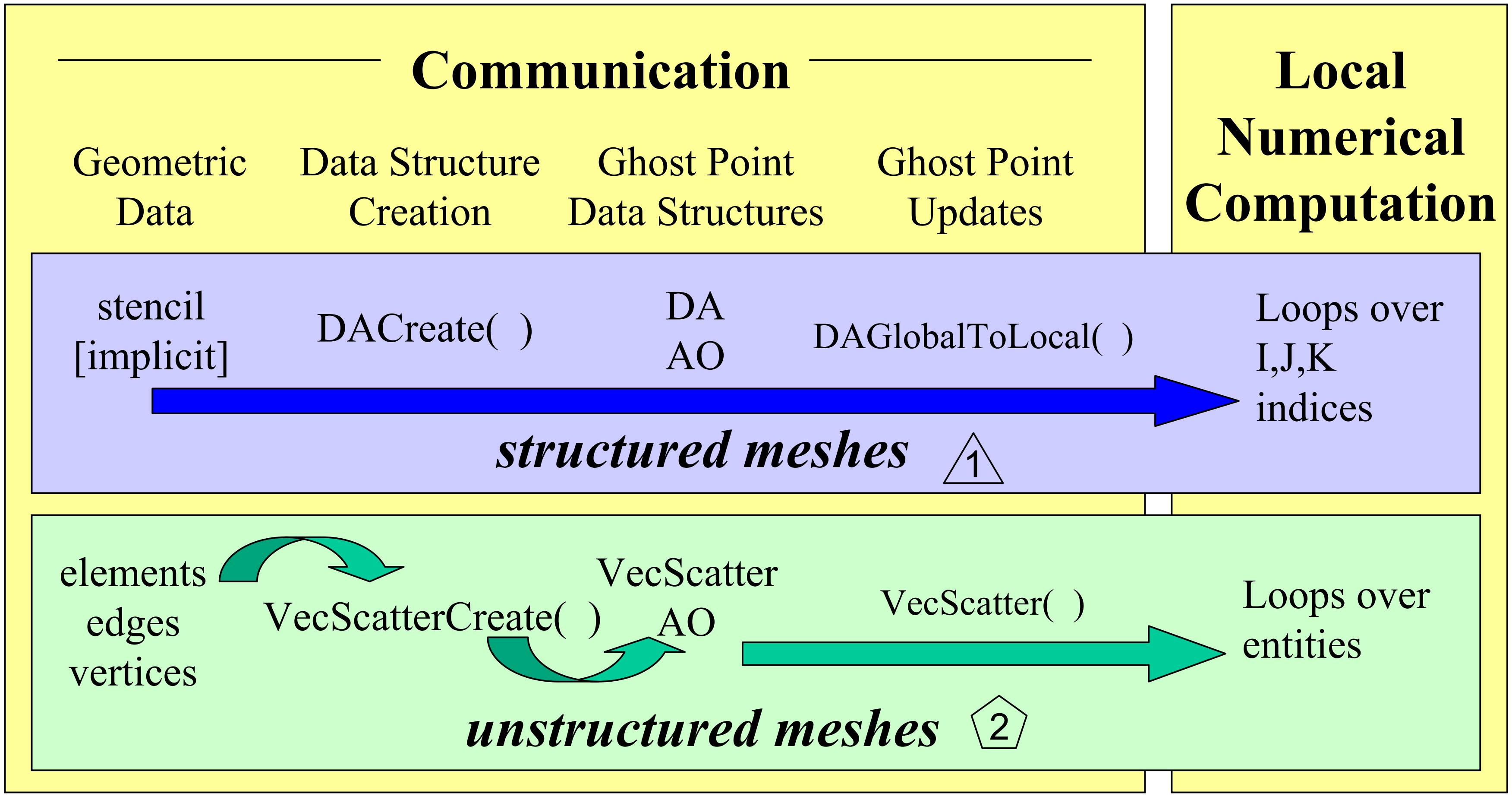
● Local node ○ Ghost node



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.

data layout

Communication and Physical Discretization



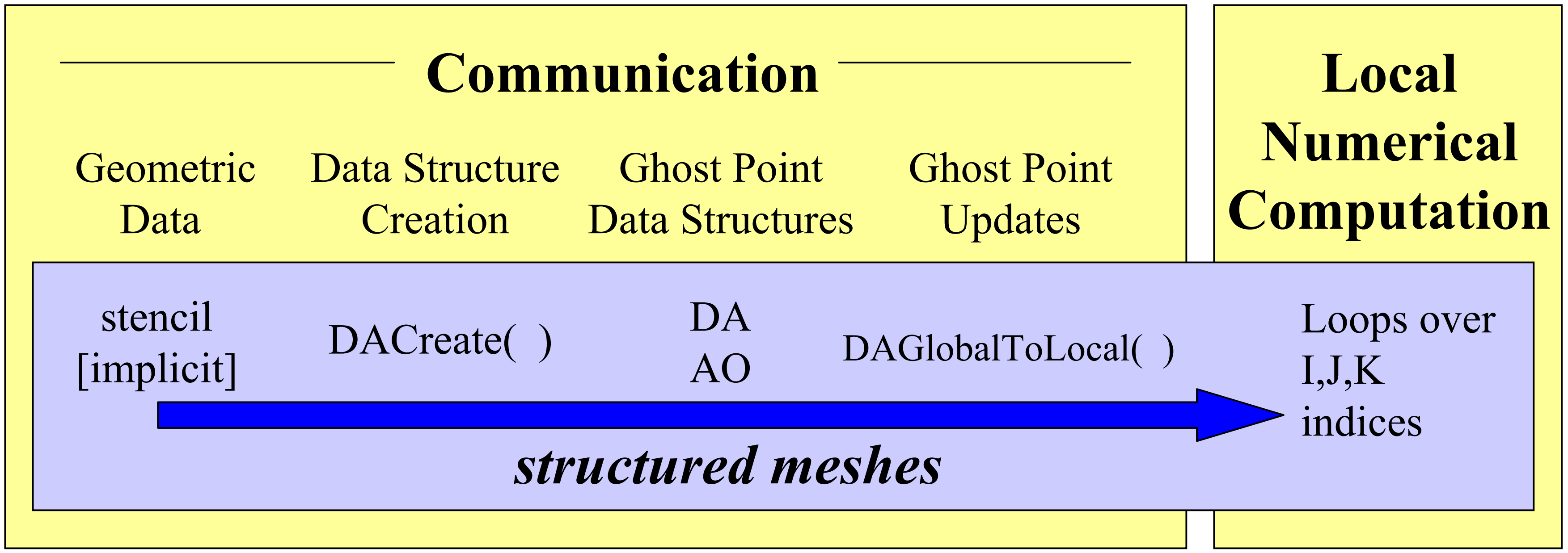
data layout

DA: Parallel Data Layout and Ghost Values for Structured Meshes

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

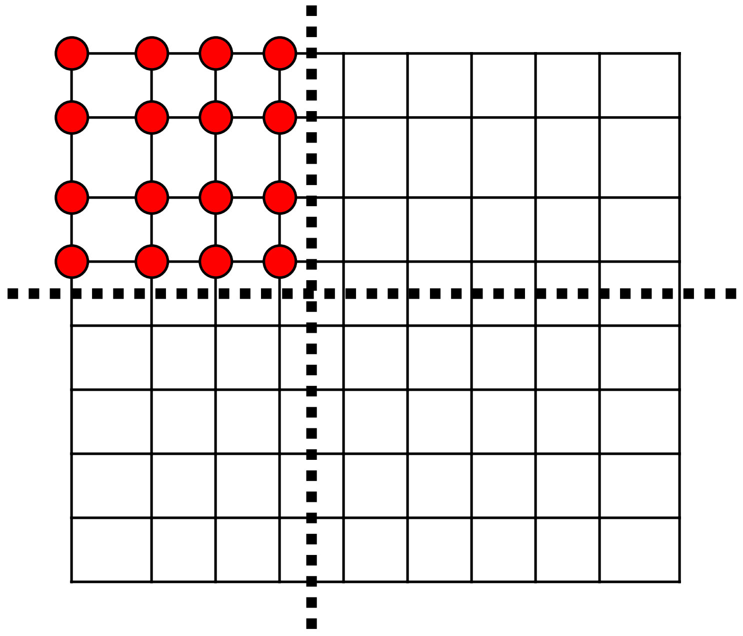
data layout:
distributed arrays

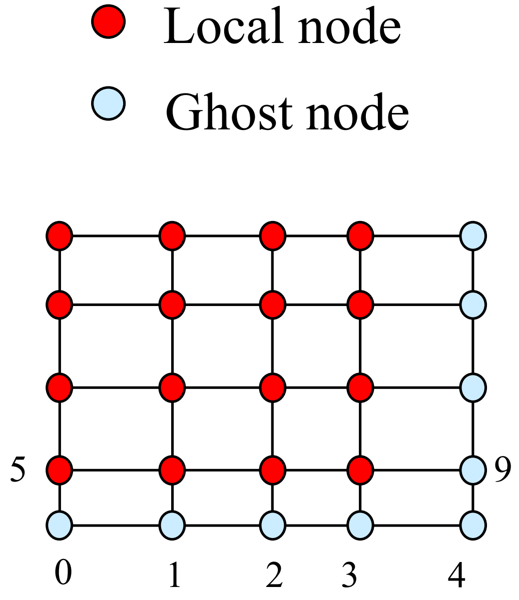
Communication and Physical Discretization: Structured Meshes



data layout:
distributed arrays

Global and Local Representations



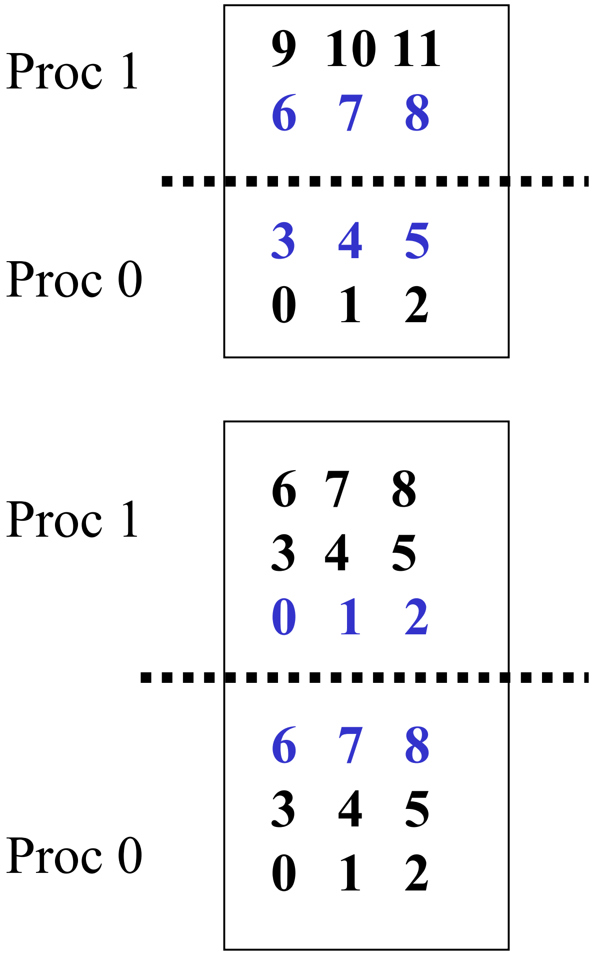


Global: each process stores a unique local set of vertices (and each vertex is owned by exactly one process)

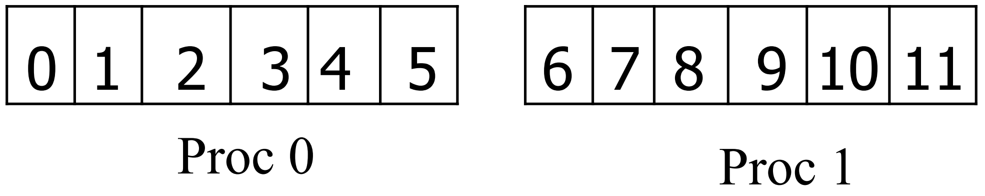
Local: each process stores a unique local set of vertices *as well as* ghost nodes from neighboring processes

data layout:
distributed arrays

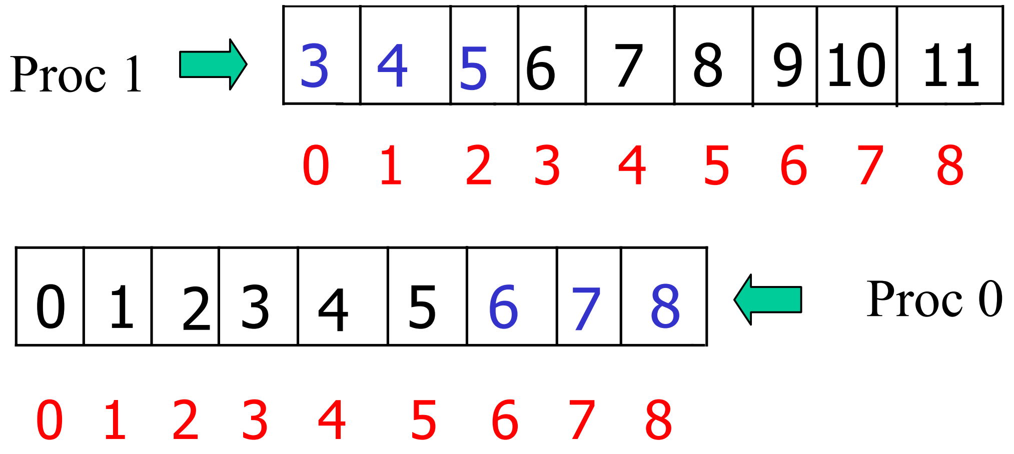
Global and Local Representations (cont.)



Global Representation:



Local Representations:



data layout:
distributed arrays

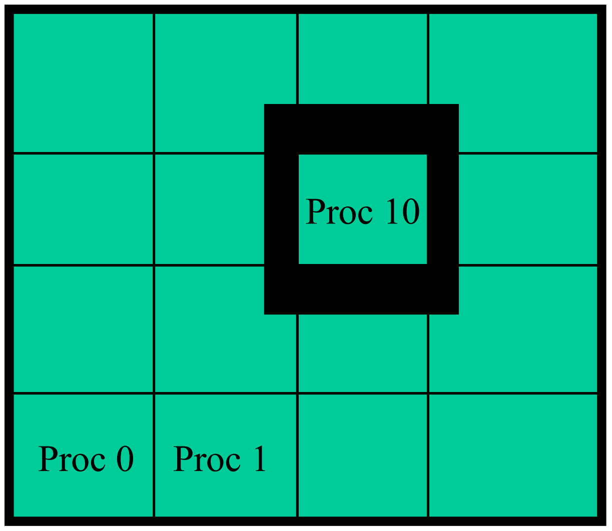
Logically Regular Meshes

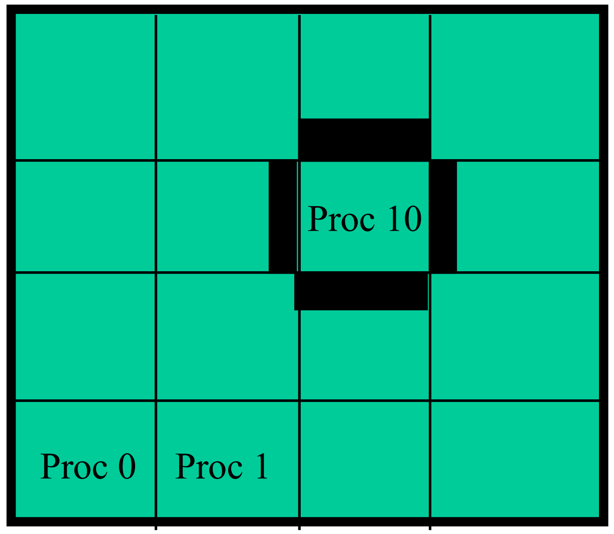
- DA - Distributed Array: object containing information about vector layout across the processes and communication of ghost values
- Form a DA
  - DACreate1d(….,DA *)
  - DACreate2d(….,DA *)
  - DACreate3d(….,DA *)
- Create the corresponding PETSc vectors
  - DACreateGlobalVector(DA, Vec *) or
  - DACreateLocalVector(DA,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

data layout:
distributed arrays

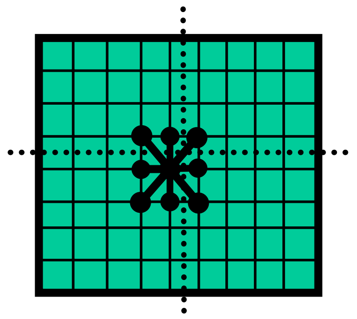
Distributed Arrays

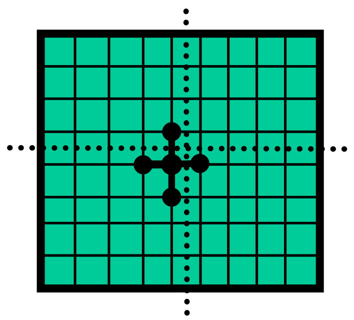
Data layout and ghost values





Box-type stencil





Star-type stencil

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 process stores its local portion plus ghost values
  - `DACreateLocalVector(DA da,Vec *lvec);`
  - uses "natural" local numbering of indices (0,1,…*nlocal*-1)

data layout:
distributed arrays

DACreate1d(…,*DA)

- DACreate1d(MPI_Comm comm,DAPeriodicType wrap,
  int M,int dof,int s,int *lc,DA *inra)
  - MPI_Comm — processes containing array
  - DA_[NONPERIODIC,XPERIODIC]
  - number of grid points in x-direction
  - degrees of freedom per node
  - stencil width
  - Number of nodes for each domain
    - Use PETSC_NULL for the default

data layout:
distributed arrays

DACreate2d(…,*DA)

- DACreate2d(MPI_Comm comm,DAPeriodicType wrap,
  DAStencilType stencil_type, int M,int N,
  int m,int n,int dof,int s,int *lx,int *ly,DA *inra)
  - DA_[NON,X,Y,XY]PERIODIC
  - DA_STENCIL_[STAR,BOX]
  - number of grid points in x- and y-directions
  - processes in x- and y-directions
  - degrees of freedom per node
  - stencil width
  - Number of nodes for each domain
    - Use PETSC_NULL for the default

data layout:
distributed arrays

Updating the Local Representation

Two-step process enables overlapping computation and communication

- DAGlobalToLocalBegin(DA, global_vec, insert,local_vec)
  - global_vec provides data
  - Insert is either INSERT_VALUES or ADD_VALUES
    - specifies how to update values in the local vector
  - local_vec is a pre-existing local vector
- DAGlobalToLocal End(DA,…)
  - Takes same arguments

data layout:
distributed arrays

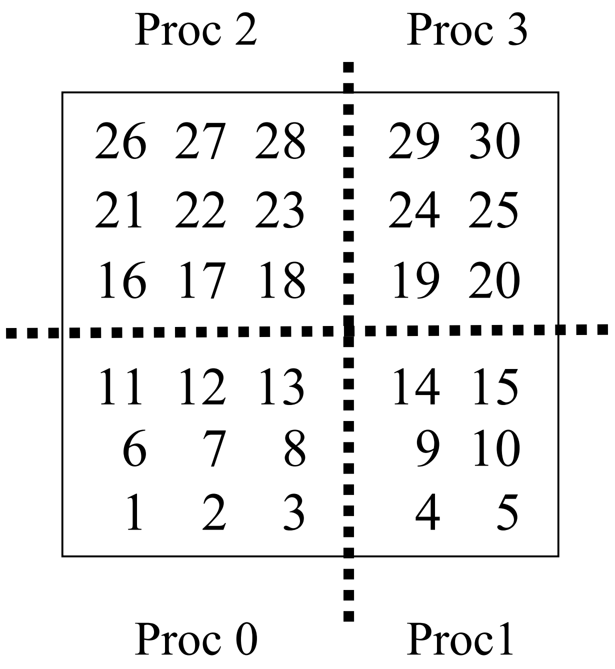
Ghost Point Scatters:

Burger’s Equation Example

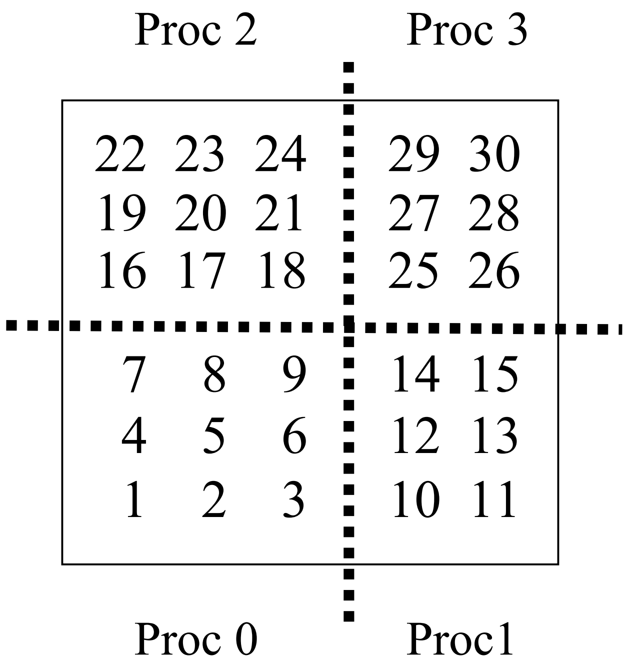
```
     call
DAGlobalToLocalBegin(da,u_global,INSERT_VALUES,ulocal,ierr)
     call
DAGlobalToLocalEnd(da,u_global,INSERT_VALUES,ulocal,ierr)

     call VecGetArray( ulocal, uv, ui, ierr )
#define u(i) uv(ui+i)
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 VecRestoreArray( ulocal, uv, ui, ierr )
     call DALocalToGlobal(da,f,INSERT_VALUES,f_globa
```

data layout:
distributed arrays

Global Numbering used by DAs



Natural numbering, corresponding to the entire problem domain



PETSc numbering used by DAs

data layout:
distributed arrays

Mapping Between Global Numberings

- Natural global numbering
  - convenient for visualization of global problem, specification of certain boundary conditions, etc.
- Can convert between various global numbering schemes using AO (Application Orderings)
  - DAGetAO(DA da, AO *ao);
  - AO usage explained in next section
- Some utilities (e.g., VecView()) automatically handle this mapping for global vectors attained from DAs

Distributed Array Example

- Files: src/snes/examples/tutorial/ex5.c, ex5f.F
  - Functions that construct vectors and matrices use a naturally associated DA
    - DAGetMatrix()
    - DASetLocalFunction()
    - DASetLocalJacobian()

data layout:
distributed arrays

The Bratu Equation I

SNES Example 5

- Create SNES and DA
- Use DASetLocalFunction()
  - Similarly DASetLocalJacobian()
- Use SNESDAFormFunction() for SNES
  - Could also use FormFunctionMatlab()
- Similarly SNESDAComputeJacobian()
  - Use DAGetMatrix() for SNES Jacobian
  - Could also use SNESDefaultComputeJacobian()

The Bratu Equation II
SNES Example 5

- int FormFunctionLocal(DALocalInfo *info,
  PetscScalar **x, PetscScalar **f,
  void *ctx)

```
for(j = info->ys; j < info->ys + info->ym; j++) {
  for(i = info->xs; i < info->xs + info->xm; i++) {
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      f[j][i] = x[j][i];
    } else {
      u       = x[j][i];
      u_xx    = -(x[j][i+1] - 2.0*u + x[j][i-1])*(hy/hx);
      u_yy    = -(x[j+1][i] - 2.0*u + x[j-1][i])*(hx/hy);
      f[j][i] = u_xx + u_yy – hx*hy*lambda*PetscExpScalar(u);
    }
  }
}
```

The Bratu Equation III
SNES Example 5

- int FormJacobianLocal(DALocalInfo *info, PetscScalar **x, Mat jac, void *ctx)

```
for(j = info->ys; j < info->ys + info->ym; j++) {
  for(i = info->xs; i < info->xs + info->xm; i++) {
    row.j = j; row.i = i;
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      v[0] = 1.0;
      MatSetValuesStencil(jac,1,&row,1,&row,v,INSERT_VALUES);
    } else {
      v[0] = -(hx/hy); col[0].j = j-1; col[0].i = i;
      v[1] = -(hy/hx); col[1].j = j;   col[1].i = i-1;
      v[2] = 2.0*(hy/hx+hx/hy) - hx*hy*lambda*PetscExpScalar(x[j][i]);
      v[3] = -(hy/hx); col[3].j = j;   col[3].i = i+1;
      v[4] = -(hx/hy); col[4].j = j+1; col[4].i = i;
      MatSetValuesStencil(jac,1,&row,5,col,v,INSERT_VALUES);
    }
  }
}
```

Interfacing to 3rd party Packages

Using PETSc with Other Packages: Linear Solvers

- Interface Approach
  - External linear solvers typically use a variant of CSR matrix
  - Each package has a matrix subclass with overridden methods
- Usage
  - Set preconditioners via the usual approach
    - Procedural interface: PCSetType(pc,"spai")
    - Runtime option: -pc_type spai
  - Set preconditioner-specific options via the usual approach
    - PCSPAISetEpsilon(), PCSPAISetVerbose(), etc.
    - -pc_spai_epsilon <eps>   -pc_spai_verbose etc.

Using PETSc with Other Packages: HYPRE - Preconditioners

- Several preconditioners
  - -pc_type hypre
  - -pc_hypre_type [pilut,parasails,boomeramg,euclid]
- Specialize preconditioner
  - -pc_hypre_boomeramg_max_levels <num>
  - -pc_hypre_boomeramg_grid_sweeps <fine,down,up,coarse>
- Options can be displayed with -help
  - Only for selected types

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)

TAO Interface

```
TAO_SOLVER     tao;              /*  optimization solver  */
Vec            x, g;             /*  solution and gradient vectors  */
ApplicationCtx usercontext;      /*  user-defined context  */

TaoInitialize();

/* Initialize Application -- Create variable and gradient vectors x and g */
...

TaoCreate(MPI_COMM_WORLD,”tao_lmvm”,&tao);
TaoSetFunctionGradient(tao,x,g, FctGrad,(void*)&usercontext);

TaoSolve(tao);

/* Finalize application -- Destroy vectors x and g */   ...

TaoDestroy(tao);
TaoFinalize();
```

Similar Fortran interface, e.g., call TaoCreate(...)

software interfacing: TAO

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: 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 allows C/Fortran programmers to use Matlab functions
    - PetscMatlabEngine – unwraps PETSc vectors and matrices for MatlabEngine
- 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)

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