Introduction to PETSc 2

Adapted from presentation by:

Matt Knepley

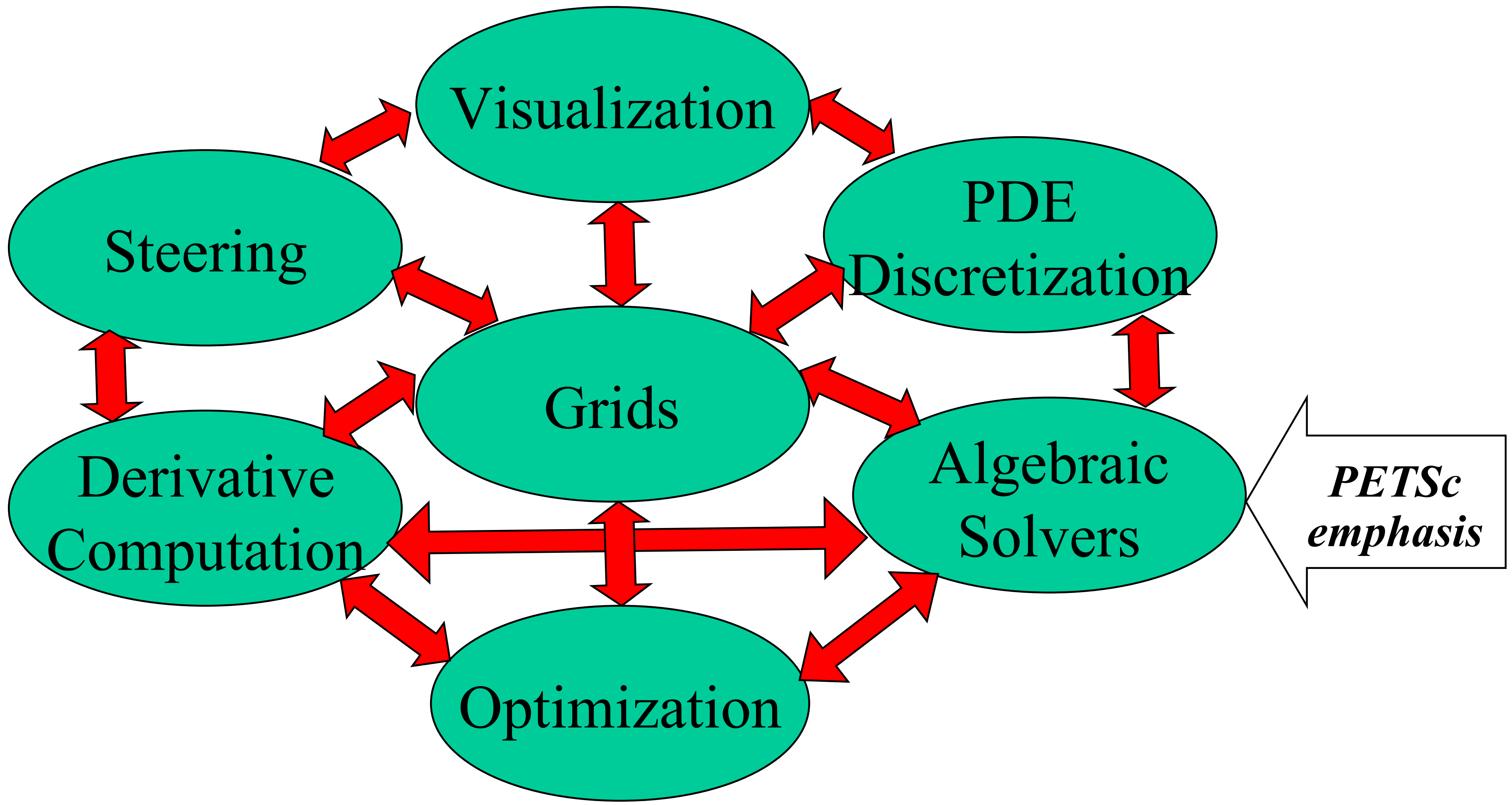
Kris Buschelman

Argonne National Laboratory

PETSc

- Writing hand-parallelized application codes from scratch is extremely difficult and time consuming.
- Scalable parallelizing compilers for real application codes are very far in the future.
- We can ease the development of parallel application codes by developing general-purpose, parallel numerical PDE libraries.
- Caveats
  - 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 reduce the development time, but it is not a black-box PDE solver nor a silver bullet.

Component Interactions for Numerical PDEs



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 funding and support
  - Department of Energy: MICS Program, DOE2000, SciDAC
  - National Science Foundation, Multidisciplinary Challenge Program, CISE

Interfaced Solvers

- 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)
- Parallel LU
  - SuperLU_DIST (Demmel and Li, LBNL)
  - SPOOLES (Ashcroft, Boeing, funded by ARPA)
  - MUMPS (European)
  - PLAPACK (van de Geijn, UT Austin)
- Parallel Cholesky
  - DSCPACK (Raghavan, Penn. State)
  - SPOOLES (Ashcroft, Boeing, funded by ARPA)
  - PLAPACK (van de Geijn, UT Austin)

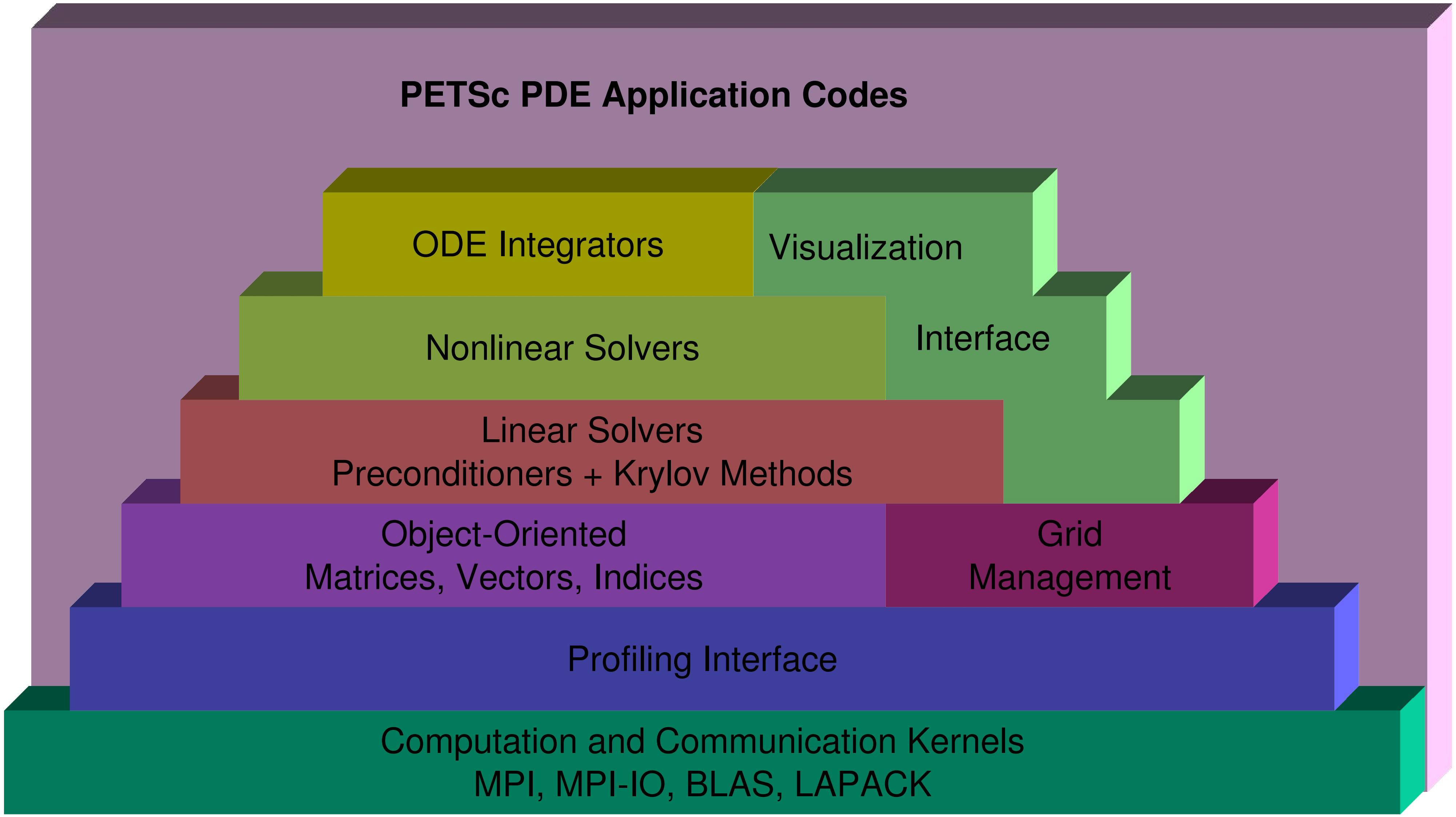
Interfaced Solvers (continued)

- XYTlib – parallel direct solver (Fischer and Tufo, ANL)
- SPAI – Sparse approximate inverse (parallel)
  - Parasails (Chow, part of Hypre, LLNL)
  - SPAI 3.0 (the Grote/Barnard implementation)
- Algebraic multigrid
  - Parallel BoomerAMG (part of Hypre, LLNL)
  - Sequential RAMG (Ruge and Stueben’s original code)
  - Sequential SAMG (Stueben’s modern version for systems of eqns)
- Parallel ICC(0) – BlockSolve95 (Jones and Plassman, ANL)
- Parallel ILU
  - BlockSolve95 (Jones and Plassman, ANL)
  - PILUT (part of Hypre, LLNL)
  - EUCLID (Hysom – also part of Hypre, ODU/LLNL)
- Sequential ILUDT (part of Saad’s SPARSEKIT2, U of MN)

Interfaced Packages

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

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

PETSc Numerical Components

Nonlinear Solvers

Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers

Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods

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

Preconditioners

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

Matrices

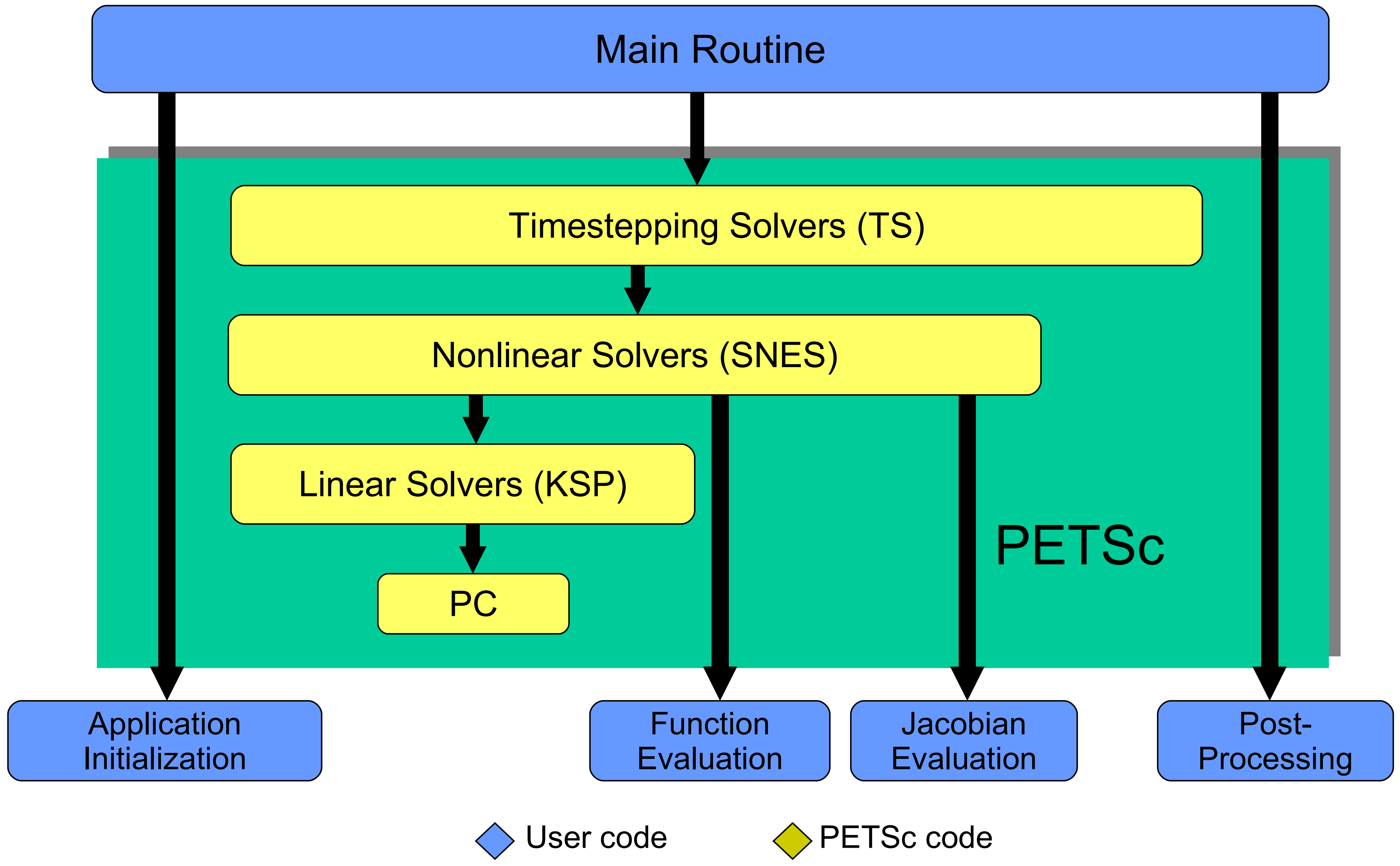
Compressed Sparse Row (AIJ)	Blocked Compressed Sparse Row (BAIJ)	Block Diagonal (BDIAG)	Dense	Other

Vectors

Index Sets

Indices	Block Indices	Stride	Other

Flow of Control for PDE Solution



Levels of Abstraction in Mathematical Software

- Application-specific interface
  - Programmer manipulates objects associated with the application
- High-level mathematics interface
  - Programmer manipulates mathematical objects
  - 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

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!

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

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

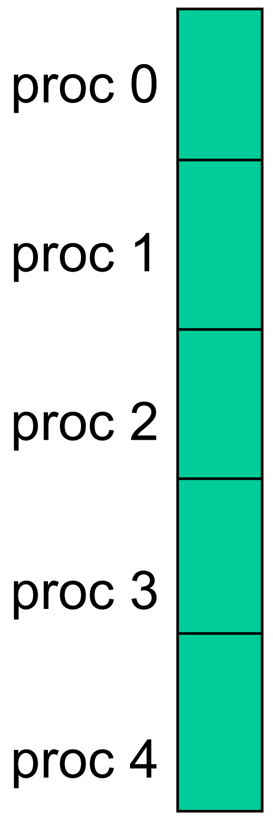
- Check for an option
  - PetscOptionsHasName()
- Retrieve a value
  - PetscOptionsGetInt(), PetscOptionsGetIntArray()
- Set a value
  - PetscOptionsSetValue()
- Clear, alias, reject, etc.

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)

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



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

How Can We Use a PETSc Vector

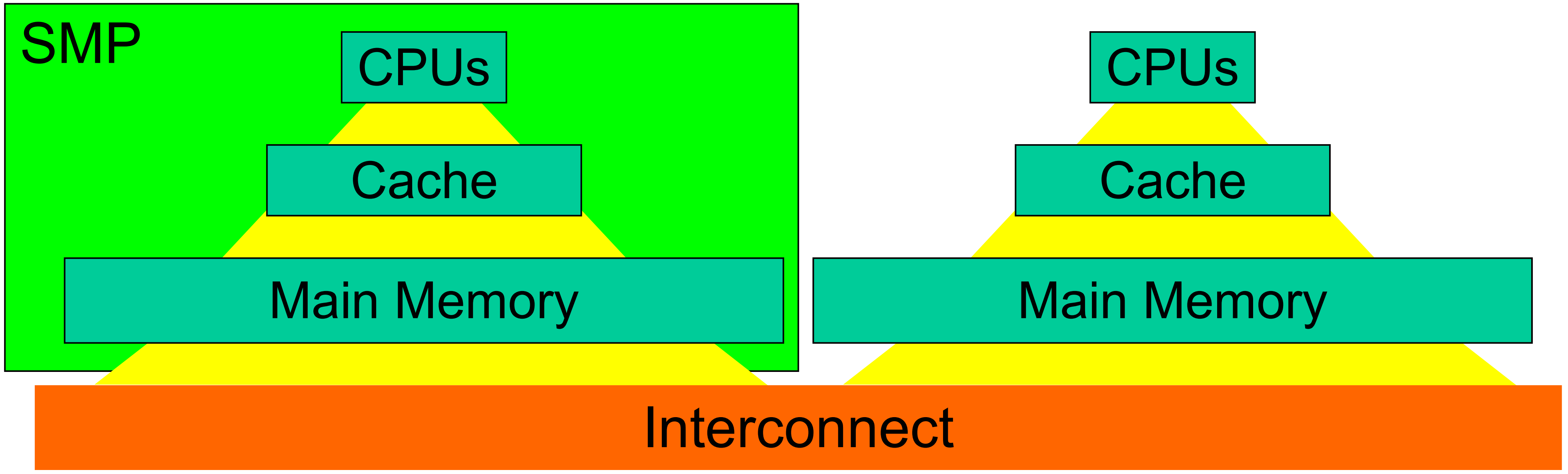
- PETSc supports “data structure-neutral” objects
  - distributed memory “shared nothing” model
  - single processors and shared memory systems
- PETSc vector is a “handle” to the real vector
  - Allows the vector to be distributed across many processes
  - To access the elements of the vector, we cannot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

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



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)

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

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

A Parallel Way to Set the Elements of A Distributed Vector

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

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, Real *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, Real *r)	$r = \|\|x\|\|$

A Complete PETSc Program

```
#include <petscvec.h>
int main(int argc,char **argv)
{
  Vec x;
  int n = 20;
  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(x,one);
  VecDot(x,x,&dot);
  PetscPrintf(PETSC_COMM_WORLD,"Vector length %d\n",(int)dot);
  VecDestroy(x);
  PetscFinalize();
  return 0;
}
```

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;
PetscScalar *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);
```

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

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.

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

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)

Matrix Assembly Example

simple 3-point stencil for 1D discretization

Choose the global size of the matrix

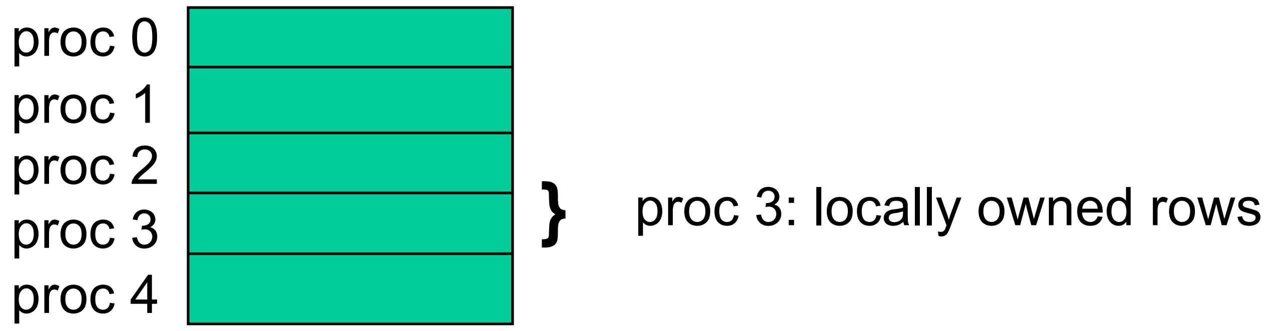
Let PETSc decide how to allocate matrix across processes

```
Mat          A;
int          column[3], i;
PetscScalar 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);
```

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

Matrix Assembly Example with Parallel Assembly

simple 3-point stencil for 1D discretization

```
Mat          A;
int          column[3], i, start, end, istart, iend;
PetscScalar 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

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.

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

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

PETSc Programming Aids

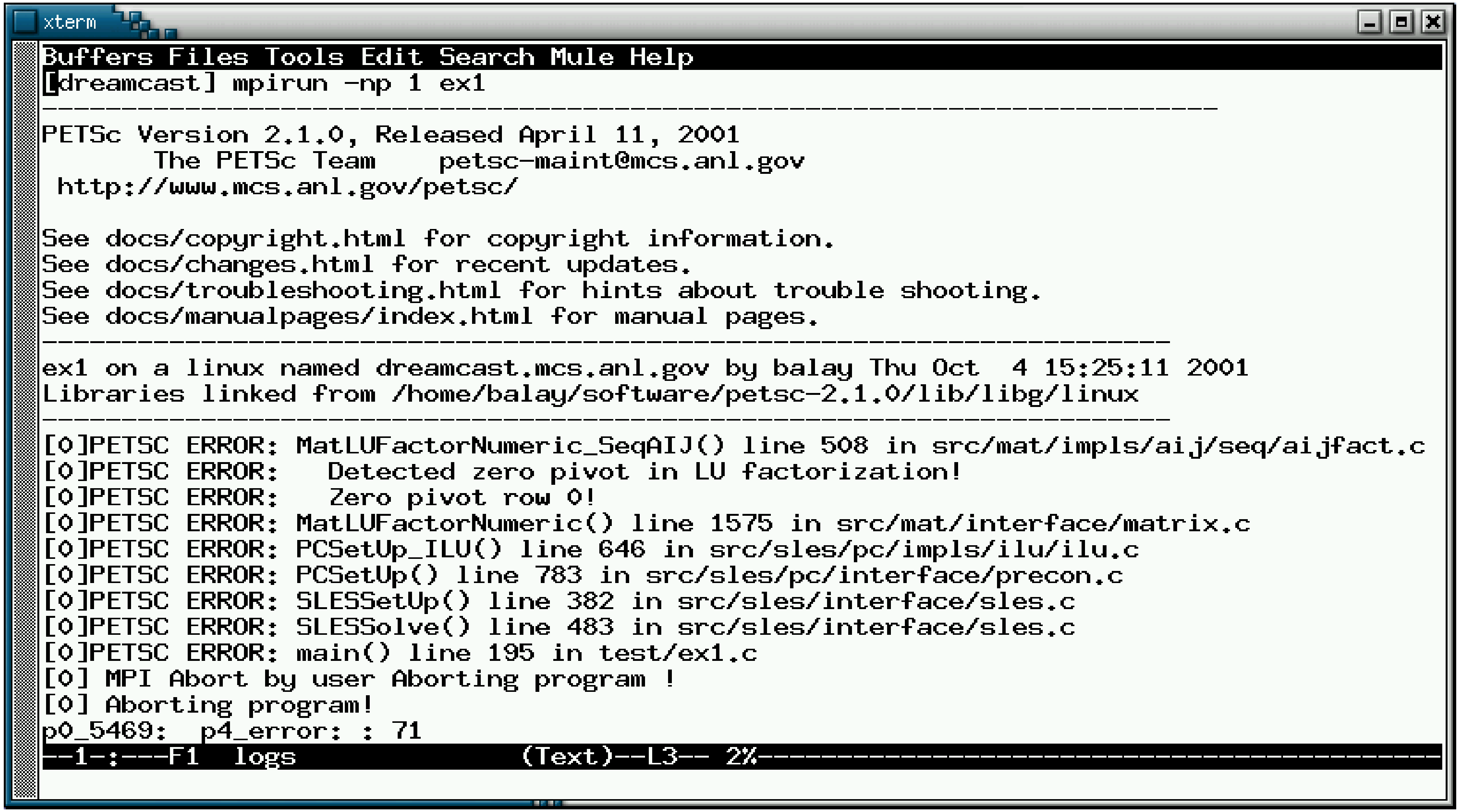
- Correctness Debugging
  - Automatic generation of tracebacks
  - Detecting memory corruption and leaks
  - Optional user-defined error handlers
- Performance Debugging
  - Integrated profiling using -log_summary
  - Profiling by stages of an application
  - User-defined events

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

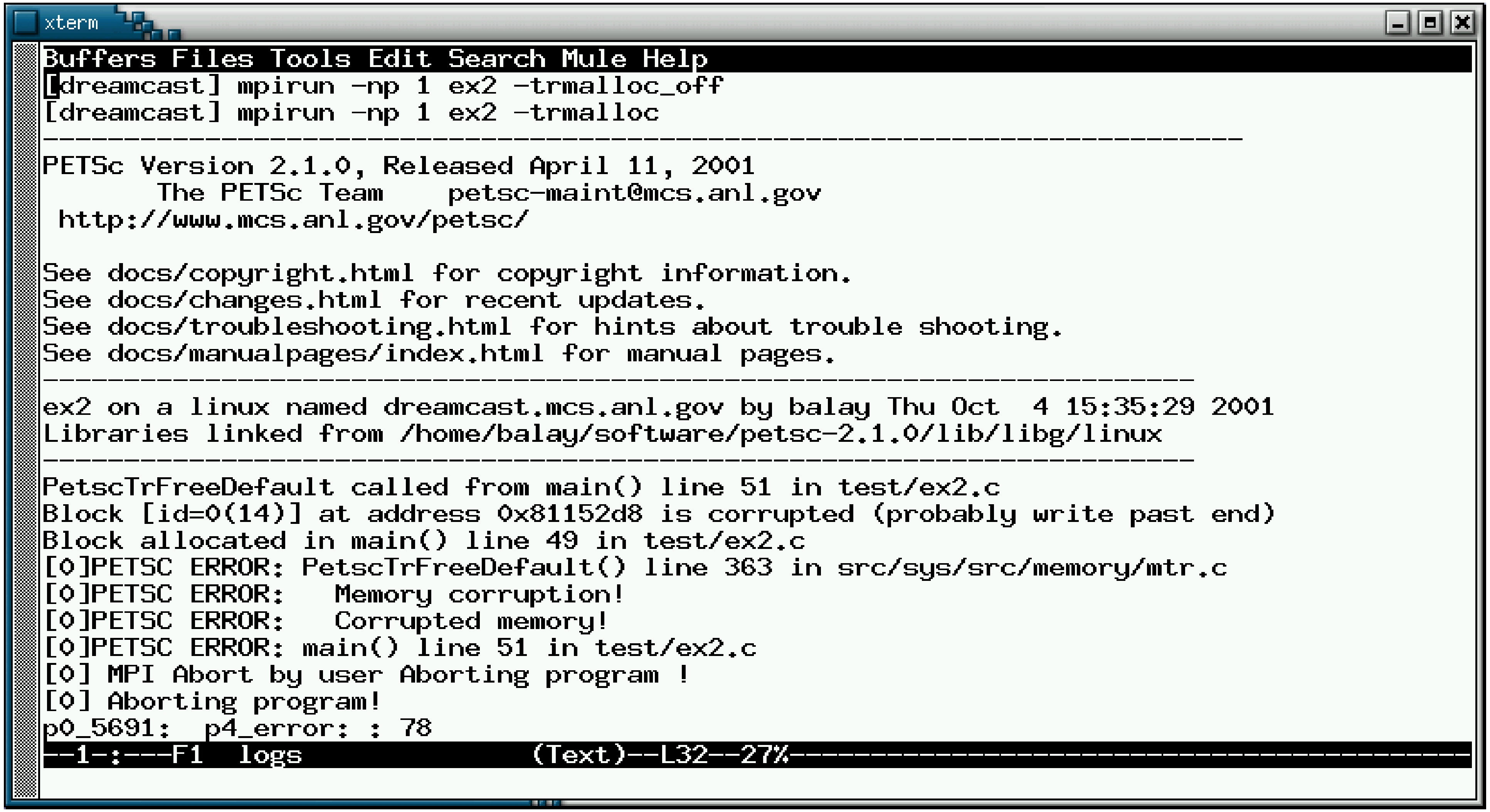
Breakdown in ILU factorization due to a zero pivot



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

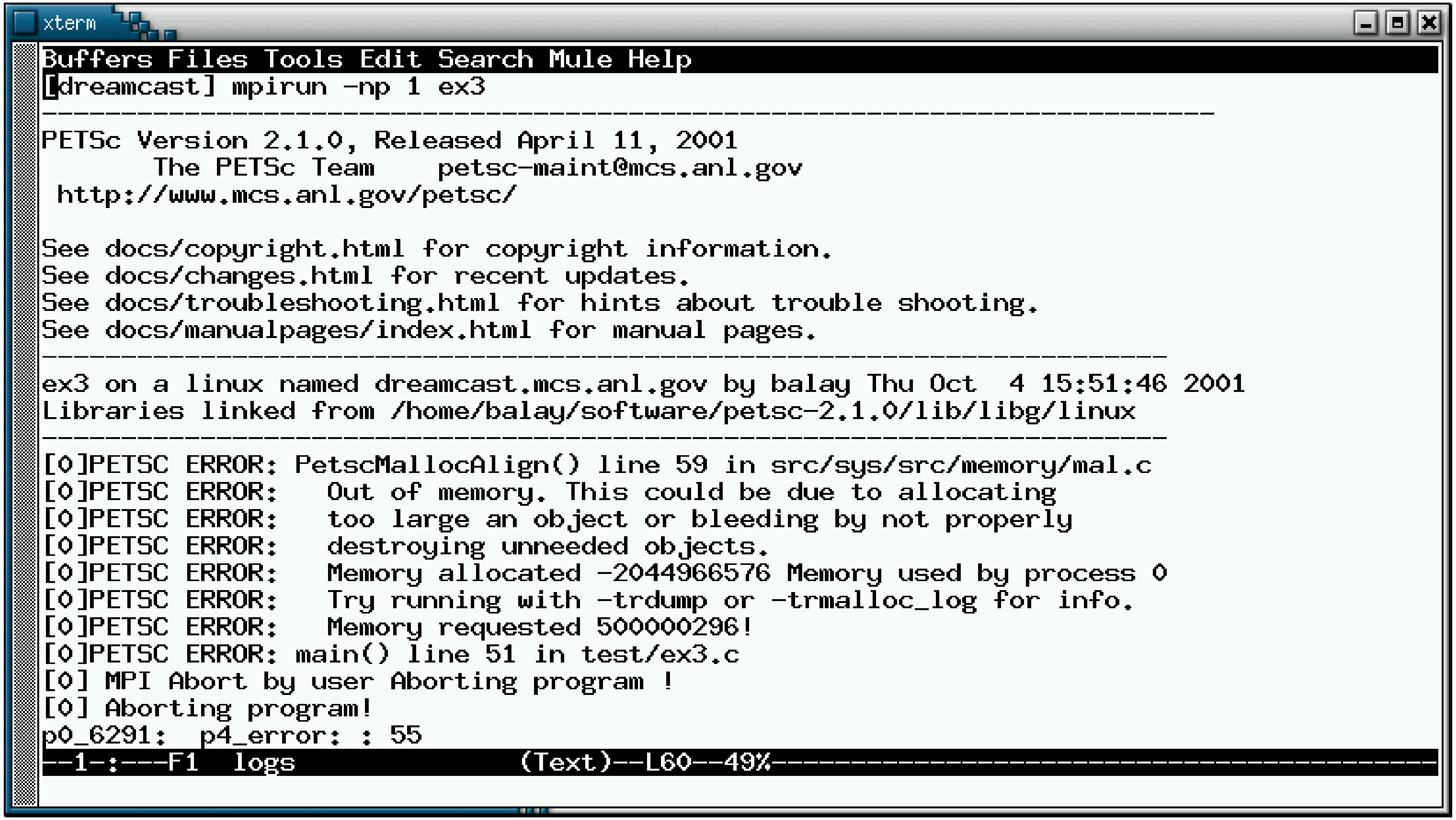
Sample Memory Corruption Error



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

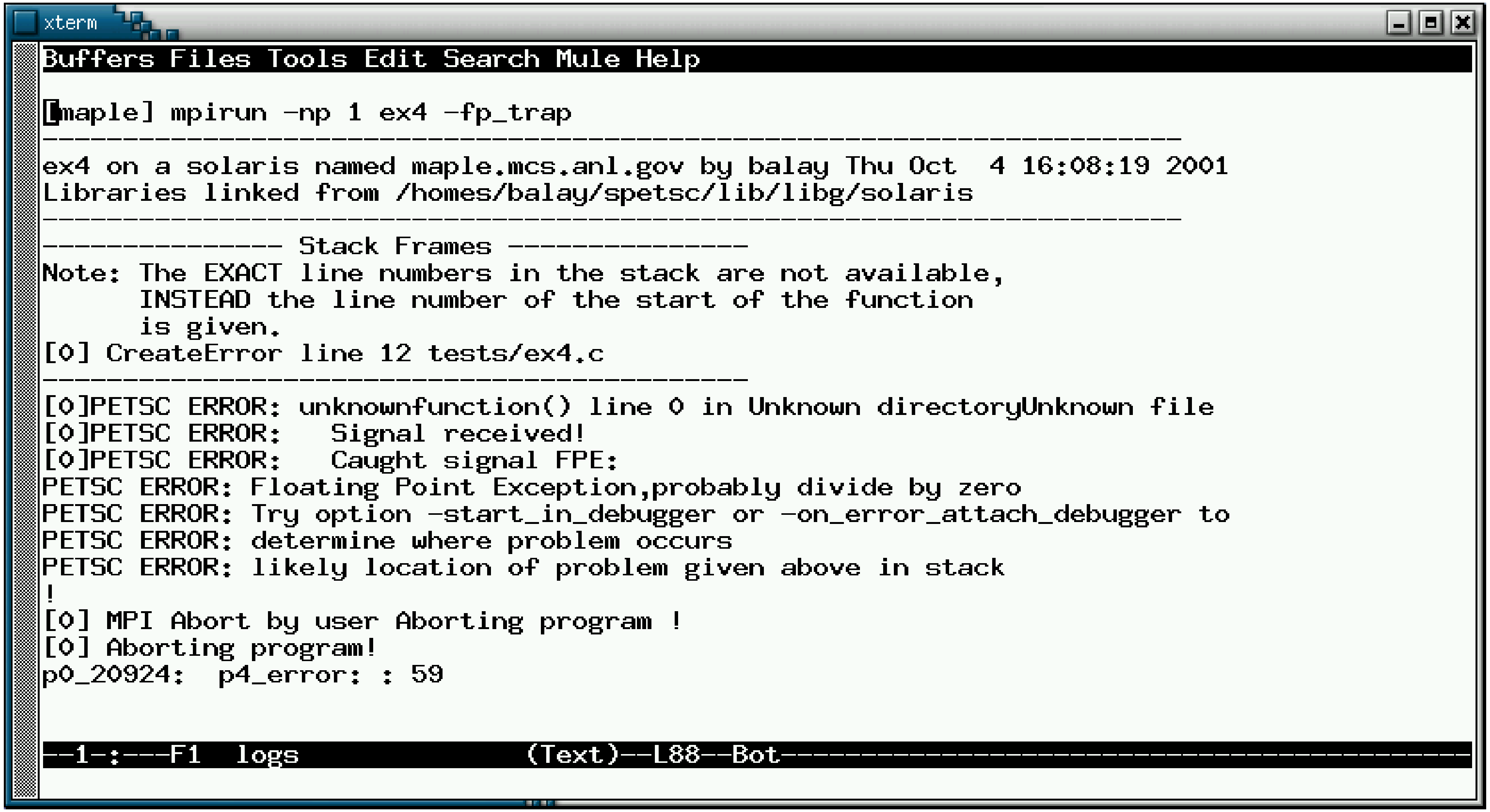
Sample Out-of-Memory Error



```
[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

- 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

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,
```

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

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

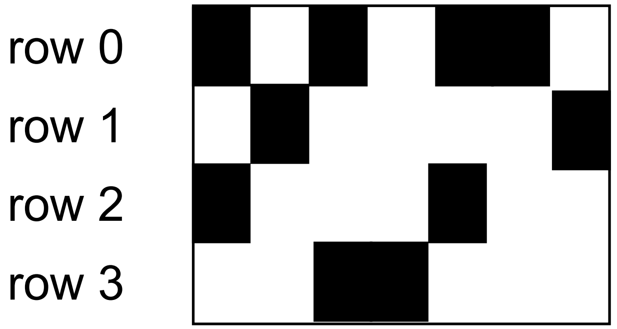
- Results from 250 MHz R10000 (500 MF/sec peak)
- BAIJ is a block AIJ with blocksize of 4
- Multiple right-hand sides can be solved in nearly the same time as a single RHS

Matrix Memory Pre-allocation

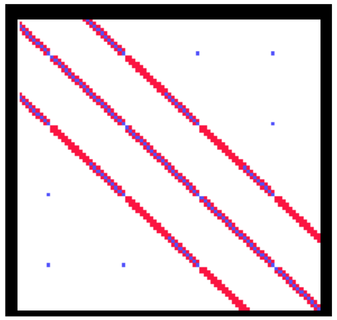
- PETSc sparse matrices are dynamic data structures. Can add additional nonzeros freely
- Dynamically adding many nonzeros
  - requires additional memory allocations
  - requires copies
  - can kill performance
- Memory pre-allocation provides the freedom of dynamic data structures plus good performance

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



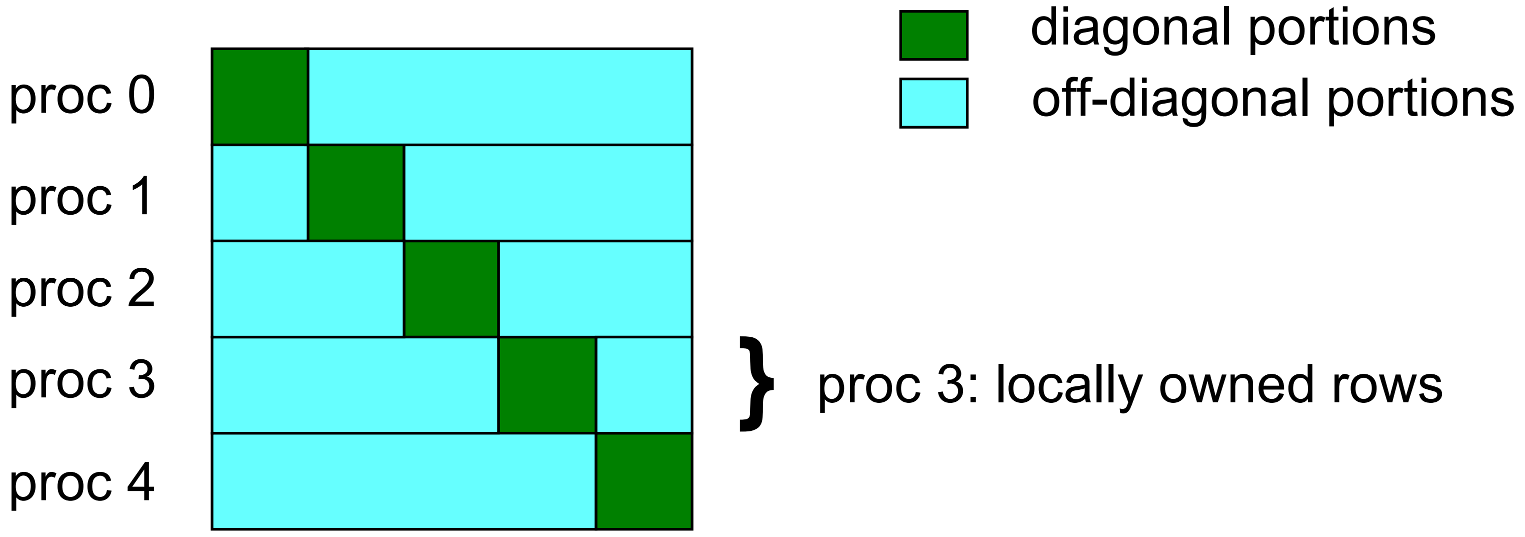
another sample nonzero pattern

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.



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!
```