PETSc Introductory Tutorial

Barry Smith
Argonne National Laboratory
June 2006
Presentation Objectives

• Who?
• When?
• Where
• Why?
• What is in PETSc?
• How to use PETSc?
quicktime and a tiff (uncompressed) decompressor are needed to see this picture.
Timeline

- **PETSc 1 release**: 1991
- **PETSc 2 release**: 1993
- **Barry** +
- **Bill** +
- **Lois** +
- **Satish** +
- **Hong** +
- **Kris** +
- **Matt** +
- **Victor** +
- **Lois** -
- **Kris** -
- **Bill** -

QuickTime® and a TIFF (Uncompressed) decompressor are needed to see this picture.
Where

Argonne: a US Dept of Energy Lab

Chicago
Why? Original Goals of PETSc

• Provide software for the **scalable** (parallel) solution of **algebraic systems** arising from partial differential equation simulations (PDEs).

• Eliminate the MPI from MPI programming
What is PETSc?

- Supported “research” code
  - 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 virtually all systems
  - Cray, SGI, IBM, HP, Sun, Linux, Windows, Apple

- Funded largely by the US Department of Energy
Features

• Support for/uses many numerical packages
• Many (parallel) vector/array operations
• Numerous (parallel) matrix formats
• **Numerous** linear solvers
• Nonlinear solvers
• Limited ODE integrators
• Parallel grid/data management
Support for/uses many numerical packages

- Packages can be automatically
  - downloaded
  - configured and built
  - installed in PETSc
Interfaced Packages

1. LU (Sequential)
   - SuperLU (Demmel and Li, LBNL)
   - ESSL (IBM)
   - Matlab
   - LUSOL (from MINOS - Michael Saunders, Stanford)
   - LAPACK
   - PLAPACK (van de Geijn, UT Austin)
   - UMFPACK (Timothy A. Davis)

   • 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 Packages

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

1. Parititioning
   - Parmetis
   - Chaco
   - Jostle
   - Party
   - Scotch
   - ODE integrators
     - Sundials (LLNL)

12. Eigenvalue solvers
   - BLOPEX (developed by Andrew Knyazev)
Size Limitations?

• PETSc has done problem with 500 million unknowns

• PETSC has used on over 6,000 processors

• PETSC applications have run at two tera-flops
Structure of PETSc

PETSc Application Codes

- ODE Integrators
- Nonlinear Solvers
- Linear Solvers
- Preconditioners + Krylov Methods
- Matrices, Vectors, Indices

PETSc Structure

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

Profiling Interface

Grid Management
Object Oriented Programming

**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 (and likely many more) are defined
The PETSc Programming Model

• Distributed memory, “shared-nothing”
  – Requires only a standard compiler
  – Access to data on remote machines through MPI

• Hide within objects the details of the communication

• User orchestrates communication at a higher abstract level then direct MPI calls
Distributed Memory Model

• Integer A(10)

  ...  

  \textbf{print} *, A

  ...  

  This A is completely different from this one
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
- 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.
What is not in PETSc?

• Higher level representations of mathematical objects
• Operator overloading
• Automatic load balancing
• Sophisticated visualization capabilities
• Optimization and sensitivity, **BUT**
Child Packages of PETSc

- **SIPs** - Shift-and-Invert Parallel Spectral Transformations
- **SLEPc** - scalable eigenvalue/eigenvector solver packages.
- **TAO** - scalable optimization algorithms

All have PETSc’s style of programming
PETSc is only a Library

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

Three Ways of Using PETSc
Getting Started

• Call PetscInitialize()
  – Sets up static data and services
  – Sets up MPI if it is not already

• Call PetscFinalize()
  – Calculates logging summary
  – Shuts down and release resources
Getting Started

PetscInitialize(&argc,&argv,(char*)0,help);
...
PetscFinalize();
return 0;

Call PetscInitialize(PETSC_NULL_CHARACTER)
...
call PetscFinalize(ierr)
end

Integration
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()
  – Automatic communication during assembly
  – Customizable communication (scatters)
# PETSc Numerical Components

## Nonlinear Solvers

<table>
<thead>
<tr>
<th>Newton-based Methods</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Search</td>
<td>Trust Region</td>
</tr>
</tbody>
</table>

## Time Steppers

<table>
<thead>
<tr>
<th>Euler</th>
<th>Backward Euler</th>
<th>Pseudo Time Stepping</th>
<th>Other</th>
</tr>
</thead>
</table>

## Krylov Subspace Methods

<table>
<thead>
<tr>
<th>GMRES</th>
<th>CG</th>
<th>CGS</th>
<th>Bi-CG-STAB</th>
<th>TFQMR</th>
<th>Richardson</th>
<th>Chebychev</th>
<th>Other</th>
</tr>
</thead>
</table>

## Preconditioners

<table>
<thead>
<tr>
<th>Additive Schwartz</th>
<th>Block Jacobi</th>
<th>Jacobi</th>
<th>ILU</th>
<th>ICC</th>
<th>LU (Sequential only)</th>
<th>Others</th>
</tr>
</thead>
</table>

## Matrices

<table>
<thead>
<tr>
<th>Compressed Sparse Row (AIJ)</th>
<th>Blocked Compressed Sparse Row (BAIJ)</th>
<th>Block Diagonal (BDIAG)</th>
<th>Dense</th>
<th>Matrix-free</th>
<th>Other</th>
</tr>
</thead>
</table>

## Distributed Arrays

## Vectors

## Index Sets

<table>
<thead>
<tr>
<th>Indices</th>
<th>Block Indices</th>
<th>Stride</th>
<th>Other</th>
</tr>
</thead>
</table>

## PETSc Numerical Components

**Distributed Arrays**

**Vectors**
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

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);

PETSc determines local size
Use PETSc to get value from command line
Global 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
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

```c
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,&i,INSERT_VALUES);
}
/* These two routines ensure that the data is distributed to the other processes */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

**data objects:** vectors

**Vector index**

**VecSetValues**

**Vector value**
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);
## Selected Vector Operations

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecAXPY(Scalar *a, Vec x, Vec y)</td>
<td>$y = y + a*x$</td>
</tr>
<tr>
<td>VecAYPX(Scalar *a, Vec x, Vec y)</td>
<td>$y = x + a*y$</td>
</tr>
<tr>
<td>VecWAXPY(Scalar *a, Vec x, Vec y, Vec w)</td>
<td>$w = a*x + y$</td>
</tr>
<tr>
<td>VecScale(Scalar *a, Vec x)</td>
<td>$x = a*x$</td>
</tr>
<tr>
<td>VecCopy(Vec x, Vec y)</td>
<td>$y = x$</td>
</tr>
<tr>
<td>VecPointwiseMult(Vec x, Vec y, Vec w)</td>
<td>$w_i = x_i *y_i$</td>
</tr>
<tr>
<td>VecMax(Vec x, int *idx, double *r)</td>
<td>$r = max x_i$</td>
</tr>
<tr>
<td>VecShift(Scalar *s, Vec x)</td>
<td>$x_i = s+x_i$</td>
</tr>
<tr>
<td>VecAbs(Vec x)</td>
<td>$x_i =</td>
</tr>
<tr>
<td>VecNorm(Vec x, NormType type , double *r)</td>
<td>$r =</td>
</tr>
</tbody>
</table>
#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;
}
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);
Indexing

- Non-trivial in parallel
- PETSc IS object, generalization of
  - \{0,3,56,9\}
  - 1:4:55
  - Indexing by block
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.
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)`

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,&A);
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,N,N);
MatSetFromOptions(A);
/* mesh interior */
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
if (rank == 0) { /* Only one process creates matrix entries */
    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
Matrix Assembly Example With Parallel Assembly

simple 3-point stencil for 1D discretization

```c
Mat A;
int column[3], i, start, end, istart, iend;
double value[3];
...
MatCreate(PETSC_COMM_WORLD,&A);
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,N,N);
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++) { /* each processor generates some of the matrix values */
    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);
```

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, for sparse matrices, is simple and makes it easier to work with other codes.
  – For applications with other ordering needs, PETSc provides “Application Orderings” (AO), described later.
Blocking: Performance Benefits

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

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

PETSc

Main Routine

Linear Solvers (KSP)

PC

Application Initialization

Evaluation of $A$ and $b$

Post-Processing

User code

PETSc code

solvers: linear

beginner
Context Variables

• Are the key to solver organization
• Contain the complete state of an algorithm, including
  – parameters (e.g., convergence tolerance)
  – functions that run the algorithm (e.g., convergence monitoring routine)
  – information about the current state (e.g., iteration number)
Creating the KSP Context

- C/C++ version
  \texttt{ierr = KSPCreate(PETSC\_COMM\_WORLD,&ksp);}

- Fortran version
  \texttt{call KSPCreate(PETSC\_COMM\_WORLD,ksp,ierr)}

- Provides an \textbf{identical} user interface for all linear solvers
  - uniprocess and parallel
  - real and complex numbers
KSP Structure

• Each KSP object actually contains two parts:
  – Krylov Space Method
    • The iterative method
    • The context contains information on method parameters (e.g., GMRES search directions), work spaces, etc
  – PC — Preconditioners
    • Knows how to apply a preconditioner
    • The context contains information on the preconditioner, such as what routine to call to apply it
Linear Solvers in PETSc

Krylov Methods (KSP)

• Conjugate Gradient
• GMRES
• CG-Squared
• Bi-CG-stab
• Transpose-free QMR
• etc.

Preconditioners (PC)

• Block Jacobi
• Overlapping Additive Schwarz
• ICC, ILU via BlockSolve95
• ILU(k), LU (direct solve, sequential only)
• Arbitrary matrix
• etc.
Basic Linear Solver Code (C/C++)

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

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

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

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

KPS   ksp
Mat    A
Vec    x, b
integer n, its, ierr

call  MatCreate( PETSC_COMM_WORLD,A,ierr )
call MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,n,n,ierr)
call  MatSetFromOptions( A, ierr )
call  VecCreate( PETSC_COMM_WORLD,x,ierr )
call  VecSetSizes( x, PETSC_DECIDE, n, ierr )
call  VecSetFromOptions( x, ierr )
call  VecDuplicate( x,b,ierr )

call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)
call  KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN,ierr)
call  KSPSetFromOptions(ksp,ierr)
call  KSPSolve(ksp,b,x,its,ierr)
call  KSPDestroy(ksp,ierr)

C - then assemble matrix and right-hand-side vector

solvers: linear
Customization Options

• Command Line Interface
  – Applies same rule to all queries via a database
  – Enables the user to have complete control at runtime, with no extra coding

• Procedural Interface
  – Provides a great deal of control on a usage-by-usage basis inside a single code
  – Gives full flexibility inside an application
Setting Solver Options at Runtime

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

1. beginner
2. intermediate

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

solvers: linear
Linear Solvers: Monitoring Convergence

- **-ksp_monitor** - Prints preconditioned residual norm
- **-ksp_xmonitor** - Plots preconditioned residual norm
- **-ksp_truemonitor** - Prints true residual norm \( \| b-Ax \| \)
- **-ksp_xtruemonitor** - Plots true residual norm \( \| b-Ax \| \)
- User-defined monitors, using callbacks

solvers: linear
Setting Solver Options within Code

- KSPSetType(KSP ksp, KSPType type)
- KSPSetTolerances(KSP ksp, PetscReal rtol, PetscReal atol, PetscReal dtol, int maxits)
- etc....

- KSPGetPC(KSP ksp, PC *pc)
  - PCSetType(PC pc, PCType)
  - PCASMSSetOverlap(PC pc, int overlap)
  - etc....
Recursion: Specifying Solvers for Schwarz Preconditioner Blocks

- Specify KSP solvers and options with “-sub” prefix, e.g.,
  - Full or incomplete factorization
    - -sub_pc_type lu
    - -sub_pc_type ilu  -sub_pc_ilu_levels <levels>
  - Can also use inner Krylov iterations, e.g.,
    - -sub_ksp_type gmres  -sub_ksp_rtol <rtol>
    - -sub_ksp_max_it <maxit>
KSP: Review of Basic Usage

- **KSPCreate( )** - Create solver context
- **KSPSetOperators( )** - Set linear operators
- **KSPSetFromOptions( )** - Set runtime solver options for [KSP,PC]
- **KSPSolve( )** - Run linear solver
- **KSPView( )** - View solver options actually used at runtime (alternative: -ksp_view)
- **KSPDestroy( )** - Destroy solver
# KSP: Review of Selected Preconditioner Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set preconditioner type</td>
<td><code>PCSetType()</code></td>
<td><code>-pc_type [lu,ilu,jacobi,sor,asm,...]</code></td>
</tr>
<tr>
<td>Set level of fill for ILU</td>
<td><code>PCILUSetLevels()</code></td>
<td><code>-pc_ilu_levels &lt;levels&gt;</code></td>
</tr>
<tr>
<td>Set SOR iterations</td>
<td><code>PCSORSGetIterations()</code></td>
<td><code>-pc_sor_its &lt;its&gt;</code></td>
</tr>
<tr>
<td>Set SOR parameter</td>
<td><code>PCSORSGetOmega()</code></td>
<td><code>-pc_sor_omega &lt;omega&gt;</code></td>
</tr>
<tr>
<td>Set additive Schwarz variant</td>
<td><code>PCASMSSetType()</code></td>
<td><code>-pc_asm_type [basic,restrict,interpolate,none]</code></td>
</tr>
<tr>
<td>Set subdomain solver options</td>
<td><code>PCGetSubKSP()</code></td>
<td><code>-sub_pc_type &lt;pctype&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>-sub_ksp_type &lt;ksptype&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>-sub_ksp_rtol &lt;rtol&gt;</code></td>
</tr>
</tbody>
</table>

And many more options...

solvers: linear: preconditioners
## Review of Selected Krylov Method Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Krylov method</td>
<td>KSPSetType( )</td>
<td>-ksp_type [cg,gmres,bcgs, tfqmr,cgs,…]</td>
</tr>
<tr>
<td>Set monitoring routine</td>
<td>KSPSetMonitor( )</td>
<td>-ksp_monitor, –ksp_xmonitor, -ksp_truemonitor, -ksp_xtruemonitor</td>
</tr>
<tr>
<td>Set convergence tolerances</td>
<td>KSPSetTolerances( )</td>
<td>-ksp_rtol &lt;rt&gt; -ksp_atol &lt;at&gt; -ksp_max_its &lt;its&gt;</td>
</tr>
<tr>
<td>Set GMRES restart parameter</td>
<td>KSPGMRESRestart( )</td>
<td>-ksp_gmres_restart &lt;restart&gt;</td>
</tr>
<tr>
<td>Set orthogonalization routine for GMRES</td>
<td>KSPGMRESOrthogonalization( )</td>
<td>-ksp_unmodifiedgramschmidt -ksp_irorthog</td>
</tr>
</tbody>
</table>

And many more options...

**beginner** | **intermediate** | **solvers: linear: Krylov methods**
Why Polymorphism?

• Programs become independent of the choice of algorithm

• Consider the question:
  – What is the best combination of iterative method and preconditioner for my problem?

• How can you answer this experimentally?
  – Old way:
  – New way:...
#!/bin/csh
#
# Sample script: Experimenting with linear solver options.
# Can be used with, e.g., petsc/src/sles/examples/tutorials/ex2.c
#
foreach np (1 2 4 8)             # number of processors
  foreach kstype (gmres bcgs tfqmr) # Krylov solver
    foreach pctype (bjacobi asm)    # preconditioner
      foreach subpcstype (jacobi sor ilu) # subdomain solver
        if ($subpcstype == ilu) then
          foreach level (0 1 2)        # level of fill for ILU(k)
            echo '****** Beginning new run ******'
            mpirun -np $np ex2 -pc_type $pctype -ksp_type $kstype \
              -sub_ksp_type preonly sub_pc_type $subpcstype \
              -sub_pc_ilu_levels $level \
              -ksp_monitor -sles_view -optionsleft
          endif
        else
          echo '****** Beginning new run ******'
          mpirun -np $np ex2 -pc_type $pctype -ksp_type $kstype \
            -sub_ksp_type preonly sub_pc_type $subpcstype \
            -ksp_monitor -sles_view -optionsleft
        endif
      end
    end
  end
end
Viewing KSP Runtime Options

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

solvers: linear

intermediate
Solve $Ax = b$

Precondition via: $M_L^{-1} A M_R^{-1} (M_R \ x) = M_L^{-1} b$

- Krylov method: Use $A$ for matrix-vector products
- Build preconditioner using either
  - $A$ - matrix that defines linear system
  - or $P$ - a different matrix (cheaper to assemble)
- KSPSetOperators(KSP ksp,
  - Mat A,
  - Mat P,
  - MatStructure flag)
Matrix-Free Solvers

- Use “shell” matrix data structure
  - `MatCreateShell(…, Mat *mfctx)`

- Define operations for use by Krylov methods
  - `MatShellSetOperation(Mat mfctx,`
    - MatOperation MATOP_MULT,
    - (void *) int (UserMult)(Mat,Vec,Vec))

- Names of matrix operations defined in `petsc/include/mat.h`

- Some defaults provided for nonlinear solver usage
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
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](http://www.mcs.anl.gov/autodiff))
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 buoyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- source code:
  ```
  petsc/src/snes/examples/tutorials/ex19.c
  ```

**Solution Components**

- velocity: $u$
- velocity: $v$
- vorticity: $\zeta$
- temperature: $T$

*Application code author: D. E. Keyes*
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, &J);
MatSetSizes(J, PETSC_DECIDE, PETSC_DECIDE, n, n)
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);
SNESolve(snes,x,&its);
SNESDestroy(snes);
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.
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;
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;
}
Sample Local Computational Loops: Driven Cavity Problem

....
for ( j = jstart; j<jend; j++ ) {
    row = (j - gys) * gxm + istart - gxs - 1;
    for ( i = istart; i<iend; i++ ) {
        row++;    u = x[row].u;
        uxx = (two * u - x[row-1].u - x[row+1].u) * hydhx;
        uyy = (two * u - x[row-gxm].u - x[row+gxm].u) * hxdhy;
        f[row].u = uxx + uyy –
                    p5 * (x[row+gxm].omega- x[row-gxm].omega) * 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 1\textsuperscript{st}-order FD
  – Dense: -snes_fd, SNESDefaultComputeJacobian()
  – Sparse via colorings: MatFDColoringCreate(), SNESDefaultComputeJacobianColor()

• Matrix-free Newton-Krylov via 1\textsuperscript{st}-order FD, no preconditioning unless specifically set by user
  – -snes_mf

• Matrix-free Newton-Krylov via 1\textsuperscript{st}-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:

- SNES_CubicLineSearch( ) - cubic line search
- SNES_QuadraticLineSearch( ) - quadratic line search
- SNES_NoLineSearch( ) - full Newton step
- SNES_NoLineSearchNoNorms( ) - full Newton step but calculates no norms (faster in parallel, useful when using a fixed number of Newton iterations instead of usual convergence testing)
- Your_Own_FavoriteLineSearchRoutine( )
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
## SNES: Review of Selected Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Procedural Interface</th>
<th>Runtime Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set nonlinear solver</td>
<td>SNESSetType( )</td>
<td>-snes_type [ls,tr,umls,umtr,…]</td>
</tr>
<tr>
<td>Set monitoring routine</td>
<td>SNESSetMonitor( )</td>
<td>-snes_monitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>–snes_xmonitor, …</td>
</tr>
<tr>
<td>Set convergence tolerances</td>
<td>SNESSetTolerances( )</td>
<td>-snes_rtol &lt;rt&gt;</td>
</tr>
<tr>
<td>Set line search routine</td>
<td>SNESSetLineSearch( )</td>
<td>-snes_atol &lt;at&gt;</td>
</tr>
<tr>
<td>View solver options</td>
<td>SNESView( )</td>
<td>-snes_max_its &lt;its&gt;</td>
</tr>
<tr>
<td>Set linear solver options</td>
<td>SLESGetSLES( )</td>
<td>-snes_eq_ls [cubic,quadratic,…]</td>
</tr>
<tr>
<td></td>
<td>SLESGetKSP( )</td>
<td>-snes_view</td>
</tr>
<tr>
<td></td>
<td>SLESGetPC( )</td>
<td>-ksp_type &lt;ksptype&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ksp_rtol &lt;krt&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-pc_type &lt;pctype&gt;</td>
</tr>
</tbody>
</table>

And many more options...
Not so Advanced PETSc
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` [gdb, dbx, noxterm]
• `-on_error_abort`
• `-debugger_nodes 0,1`
• `-display machinename:0.0`

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

Breakdown in ILU factorization due to a zero pivot

depth [xterm buffer]

 Buffers Files Tools Edit Search Mule Help
 [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.

Libraries linked from /home/balay/software/petsc-2.1.0/lib/libx/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/lu/lu.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%---

depth [xterm buffer]

debugging and errors
Sample Memory Corruption Error

```
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.

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
[O]PETSC ERROR: PetscTrFreeDefault() line 363 in src/sys/src/memory/mtr.c
[O]PETSC ERROR: Memory corruption!
[O]PETSC ERROR: Corrupted memory!
[O]PETSC ERROR: main() line 51 in test/ex2.c
[O] MPI Abort by user Aborting program!
[O] Aborting program!
p0_5691: _p4_error: : 78
```
Sample Out-of-Memory Error

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.

Libraries linked from /home/balay/software/petsc-2.1.0/lib/libg/linux

PETSC ERROR: PetscMallocAlign() line 59 in src/sys/src/memory/mal.c
PETSC ERROR: Out of memory. This could be due to allocating
PETSC ERROR: too large an object or bleeding by not properly
PETSC ERROR: destroying unneeded objects.
PETSC ERROR: Memory allocated -2044966576 Memory used by process 0
PETSC ERROR: Try running with -trdump or -trmalloc_log for info.
PETSC ERROR: Memory requested 5000000296!
PETSC ERROR: main() line 51 in test/ex3.c
MPI Abort by user Aborting program!
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 directory
PETSC ERROR: Signal received!
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 Abort by user
[0] Aborting program!
[0] Aborting program!
p0_20924: p4_error: : 59
```
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

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Time (sec)</th>
<th>Flops/sec</th>
<th>--- Global ---</th>
<th>--- Stage ---</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Ratio</td>
<td>Max Ratio</td>
<td>%T %F %M %L %R</td>
<td>%T %F %M %L %R</td>
<td>Mflop/s</td>
</tr>
<tr>
<td>PetscBarrier</td>
<td>2</td>
<td>1.1733e-05</td>
<td>1.0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>VecSet</td>
<td>2</td>
<td>9.3448e-04</td>
<td>1.0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>MatMultTranspose</td>
<td>1</td>
<td>1.8022e-03</td>
<td>1.85e+08</td>
<td>0 0 0 0 57 0</td>
<td>0 0 0 0 185</td>
<td>185</td>
</tr>
<tr>
<td>MatAssemblyBegin</td>
<td>3</td>
<td>1.0057e-05</td>
<td>1.0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>MatAssemblyEnd</td>
<td>3</td>
<td>2.0356e-02</td>
<td>2.0356e-02</td>
<td>0 0 0 0 5 0</td>
<td>0 0 0 0 185</td>
<td>0</td>
</tr>
<tr>
<td>MatFDColorCreate</td>
<td>2</td>
<td>1.5341e-01</td>
<td>1.8022e-03</td>
<td>0 0 0 0 16 36</td>
<td>0 0 0 0 74 0</td>
<td>0</td>
</tr>
<tr>
<td>VecDot</td>
<td>2</td>
<td>3.2985e-03</td>
<td>9.56e+07</td>
<td>0 0 0 0 2 96</td>
<td>0 0 0 0 2 96</td>
<td>96</td>
</tr>
<tr>
<td>VecMDot</td>
<td>45</td>
<td>9.3093e-02</td>
<td>1.59e+08</td>
<td>0 0 0 0 5 1 1</td>
<td>0 0 0 0 19 159</td>
<td>159</td>
</tr>
<tr>
<td>VecNorm</td>
<td>112</td>
<td>2.0851e-01</td>
<td>8.47e+07</td>
<td>0 0 0 0 64 85</td>
<td>0 0 0 0 64 85</td>
<td>85</td>
</tr>
</tbody>
</table>

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

---

--- Event Stage 0: Main Stage

--- Event Stage 1: SetUp

--- Event Stage 2: Solve

---

--- Event Stage 1: SetUp

--- Event Stage 2: Solve
More –log_summary

Memory usage is given in bytes:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Creations</th>
<th>Destructions</th>
<th>Memory</th>
<th>Descendants' Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>--- Event Stage 0: Main Stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed array</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2.37475e+06</td>
</tr>
<tr>
<td>Index Set</td>
<td>104</td>
<td>24</td>
<td>2376480</td>
<td>0</td>
</tr>
<tr>
<td>Map</td>
<td>40</td>
<td>10</td>
<td>2000</td>
<td>0</td>
</tr>
<tr>
<td>Vec</td>
<td>36</td>
<td>10</td>
<td>2846384</td>
<td>0</td>
</tr>
<tr>
<td>Vec Scatter</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IS Local to global mapping</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Matrix</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Matrix FD Coloring</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SNES</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Krylov Solver</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Preconditioner</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>--- Event Stage 1: SetUp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed array</td>
<td>0</td>
<td>4</td>
<td>822496</td>
<td>3.16488e+06</td>
</tr>
<tr>
<td>Index Set</td>
<td>20</td>
<td>100</td>
<td>3578544</td>
<td>0</td>
</tr>
<tr>
<td>Map</td>
<td>26</td>
<td>56</td>
<td>11200</td>
<td>0</td>
</tr>
<tr>
<td>Vec</td>
<td>160</td>
<td>186</td>
<td>92490656</td>
<td>2864</td>
</tr>
<tr>
<td>Vec Scatter</td>
<td>0</td>
<td>12</td>
<td>2374784</td>
<td>0</td>
</tr>
</tbody>
</table>
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
cc-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

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

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

profiling and performance tuning
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);

profiling and performance tuning
Adding a new Class

- Cookie identifies a class uniquely

```c
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:
  ```c
  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:**
  - $\text{nnz (sizeof(double) + sizeof(int)) + n (2 \times \text{sizeof(double)} + \text{sizeof(int)})}$
  - Perfect cache (never load same data twice)

- **Computation**
  - $\text{nnz multiply-add (MA)}$

- **Roughly 12 bytes per MA**

- **Typical WS node can move $\frac{1}{2}$-4 bytes/MA**
  - *Maximum* performance is 4-33% of peak
More Performance Analysis

• Instruction Counts:
  – nnz \((2 \times \text{load-double} + \text{load-int} + \text{mult-add}) + n (\text{load-int} + \text{store-double})\)

• 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:

<table>
<thead>
<tr>
<th>Format</th>
<th>Number of Vectors</th>
<th>Mflops Ideal</th>
<th>Mflops Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIJ</td>
<td>1</td>
<td>49</td>
<td>45</td>
</tr>
<tr>
<td>AIJ</td>
<td>4</td>
<td>182</td>
<td>120</td>
</tr>
<tr>
<td>BAIJ</td>
<td>1</td>
<td>64</td>
<td>55</td>
</tr>
<tr>
<td>BAIJ</td>
<td>4</td>
<td>236</td>
<td>175</td>
</tr>
</tbody>
</table>

- 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

profiling and performance tuning
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

row 0
row 1
row 2
row 3

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.
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

profiling and performance tuning
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
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 3\textsuperscript{rd} 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 `PetscFLListAdd()`
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
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*
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.
Communication and Physical Discretization

- Geometric Data
- Data Structure Creation
- Ghost Point Data Structures
- Ghost Point Updates

**Communication**
- stencil [implicit]
- DACreate( )
- DA AO
- DAGlobalToLocal( )

**Structured Meshes**
- loops over I,J,K indices

**Unstructured Meshes**
- elements
- edges
- vertices
- VecScatterCreate( )
- VecScatter AO
- VecScatter( )

**Local Numerical Computation**
- Loops over entities

**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

Communication
- Geometric Data
- Data Structure Creation
- Ghost Point Data Structures
- Ghost Point Updates

Local Numerical Computation
- Loops over I,J,K indices

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:

Proc 0

0 1 2 3 4 5
6 7 8 9 10 11

Proc 1

0 1 2 3 4 5
6 7 8 9 10 11

Local Representations:

Proc 1

0 1 2 3 4 5
6 7 8 9 10 11

Proc 0

0 1 2 3 4 5
6 7 8 9 10 11

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, …)
Distributed Arrays

Data layout and ghost values

**Box-type stencil**

**Star-type stencil**
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)
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
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
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
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_global,ierr)

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()
The Bratu Equation I
SNES Example 5

• Create SNES and DA
• Use DASetLocalFunction()
  – Similarly DASetLocalJacobian()
• Use SNESDADAFormFunction() for SNES
  – Could also use FormFunctionMatlab()
• Similarly SNESDADAComputeJacobian()
  – 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);
        }
    }
}

VecScatter: Vector Scatters & Gathers

Parallel data layout and ghost values for unstructured meshes

- Mapping between global numberings
- Local and global indices
- VecScatter creation
- Ghost point updates
- Setting object data using local indices
Unstructured Meshes

• Setting up communication patterns is much more complicated than the structured case due to
  – mesh dependence
  – discretization dependence
Sample Differences Among Discretizations

- Cell-centered
- Vertex-centered
- Cell and vertex centered (e.g., staggered grids)
- Mixed triangles and quadrilaterals
Unstructured Mesh Concepts

• AO: Application Orderings
  – map between various global numbering schemes

• IS: Index Sets
  – indicate collections of nodes, cells, etc.

• VecScatter:
  – update ghost points using vector scatters/gathers

• ISLocalToGlobalMapping
  – map from a local (on-processor) numbering of indices to a global (across-processor) numbering
Communication and Physical Discretization: Unstructured Meshes

Communication

Geometric Data  Data Structure Creation  Ghost Point Data Structures  Ghost Point Updates

Loops over entities

intermediate

Local Numerical Computation

data layout: vector scatters

VecScatterCreate(  )
VecScatterAO
VecScatter(  )
Setting Up the Communication Pattern
(the steps in creating VecScatter)

- Renumber objects so that contiguous entries are adjacent (AO)
- Determine needed neighbor values
- Generate local numbering
- Generate local and global vectors
- Create communication object (VecScatter)
Cell-based Finite Volume Application Numbering

global indices defined by application

Proc 0

1  2
0  5
11 10

Proc 1

3  4
6  7
9  8

data layout: vector scatters

intermediate
PETSc Parallel Numbering

Proc 0

0 1
2 3
4 5

Proc 1

6 7
8 9
10 11

global indices numbered contiguously on each processor

data layout: vector scatters
Remapping Global Numbers: An Example

• Processor 0
  – nlocal: 6
  – app_numbers: \{1,2,0,5,11,10\}
  – petsc_numbers: \{0,1,2,3,4,5\}

• Processor 1
  – n_local: 6
  – app_numbers: \{3,4,6,7,9,8\}
  – petsc_numbers: \{6,7,8,9,10,11\}
Remapping Numbers (1)

- Generate a parallel object (AO) to use in mapping between numbering schemes
- AOCreateBasic(MPI_Comm comm, 
  - int nlocal, 
  - int *app_numbers, 
  - int *petsc_numbers, 
  - AO *ao);
Remapping Numbers (2)

• AOApplicationToPetsc(AO *ao,
  – int number_to_convert,
  – int *indices);

• For example, if indices[] contains the cell neighbor lists in an application numbering, then apply AO to convert to new numbering
Neighbors

using global PETSc numbers

Proc 0

0 1 6
2 3 8
4 5 10

Proc 1

1 6 7
3 8 9
5 10 11

ghost cells

intermediate

data layout: vector scatters
Local Numbering

Proc 0

0 1 6
2 3 7
4 5 8

Proc 1

6 0 1
7 2 3
8 4 5

ghost cells

intermediate

data layout: vector scatters
Global and Local Representations

• Global representation
  – parallel vector with no ghost locations
  – suitable for use by PETSc parallel solvers

• Local representation
  – sequential vectors with room for ghost points
  – used to evaluate functions, Jacobians, etc.
Global and Local Representations

Global representation:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Local Representations:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Data layout: vector scatters
Creating Vectors

- **Sequential**
  
  ```c
  VecCreateSeq(PETSC_COMM_SELF,
               9,Vec *lvec);
  ```

- **Parallel**
  
  ```c
  VecCreateMPI(PETSC_COMM_WORLD,
               6,PETSC_DETERMINE,Vec *gvec)
  ```
Local and Global Indices

• Processor 0
  – ISCreateGeneral(PETSC_COMM_WORLD, 9,{0,1,2,3,4,5,6,8,10}, IS *isg);
  – ISCreateStride(PETSC_COMM_SELF, 9,0,1,IS* isl);

• Processor 1
  – ISCreateGeneral(PETSC_COMM_WORLD, 9,{6,7,8,9,10,11,1,3,5}, IS *isg);
  – ISCreateStride(PETSC_COMM_SELF, 9,0,1,IS* isl);

ISCreatenGeneral() - Specify global numbers of locally owned cells, including ghost cells
ISCreatenStride() - Specify local numbers of locally owned cells, including ghost cells

Data layout: vector scatters
Creating Communication Objects

- VecScatterCreate(Vec gvec,
  - IS gis,
  - Vec lvec,
  - IS lis
  - VecScatter gtol);

- Determines all required messages for mapping data from a global vector to local (ghosted) vectors
Performing a Global-to-Local Scatter

Two-step process that enables overlapping computation and communication

- VecScatterBegin(VecScatter gtol,
  - Vec gvec,
  - Vec lvec,
  - INSERT_VALUES
  - SCATTER_FORWARD);
- VecScatterEnd(...);
Performing a Local-to-Global Scatter

- \texttt{VecScatterBegin(VecScatter gtol,}
  - \texttt{Vec lvec,}
  - \texttt{Vec gvec,}
  - \texttt{ADD\_VALUES,}
  - \texttt{SCATTER\_REVERSE);}\
- \texttt{VecScatterEnd(...);}
Setting Values in Global Vectors and Matrices using a Local Numbering

- Create mapping
  - ISLocalToGlobalMappingCreateIS(IS gis, ISLocalToGlobalMapping *lgmap);
- Set mapping
  - VecSetLocalToGlobalMapping(Vec gvec, ISLocalToGlobalMapping lgmap);
  - MatSetLocalToGlobalMapping(Mat gmat, ISLocalToGlobalMapping lgmap);
- Set values with local numbering
  - VecSetValuesLocal(Vec gvec, int ncols, int localcolumns, Scalar *values, ...);
  - MatSetValuesLocal(Mat gmat, ...);
  - MatSetValuesLocalBlocked(Mat gmat, ...);
**Sample Function Evaluation**

```c
int FormFunction(SNES snes, Vec Xglobal, Vec Fglobal, void *ptr)
{
    AppCtx *user = (AppCtx *) ptr;
    double x1, x2, f1, f2, *x, *f;
    int *edges = user->edges;

    VecScatterBegin(user->scatter, Xglobal, Xlocal, SCATTER_FORWARD, INSERT_VALUES);
    VecScatterEnd(user->scatter, Xglobal, Xlocal, SCATTER_FORWARD, INSERT_VALUES);

    VecGetArray(Xlocal,&X);   VecGetArray(Flocal,&F);
    for (i=0; i < user->nlocal; i++)  {
        x1 = X[edges[2*i]];  x2 = X[edges[2*i+1]];  /* then compute f1, f2 */
        F[edges[2*i]] += f1;  F[edges[2*i+1]] += f2;
    }
    VecRestoreArray(Xlocal,&X);   VecRestoreArray(Flocal,&F);

    VecScatterBegin(user->scatter, Flocal, Fglobal, SCATTER_REVERSE, INSERT_VALUES);
    VecScatterEnd(user->scatter, Flocal, Fglobal, SCATTER_REVERSE, INSERT_VALUES);
    return 0;
}
```

**Data Layout:**
- **vector scatters**

**Intermediate:**
- `VecScatterBegin` and `VecScatterEnd` for moving data between `Xlocal` and `Xglobal`.
- A loop to compute `f1` and `f2`.
- `VecGetArray` and `VecRestoreArray` for accessing and restoring arrays.
- `VecScatterBegin` and `VecScatterEnd` for moving data between `Flocal` and `Fglobal`.
Communication and Physical Discretization

**Communication**

- Geometric Data
- Data Structure Creation
- Ghost Point Data Structures
- Ghost Point Updates

**Local Numerical Computation**

- Loops over I,J,K indices

**structured mesh**

- stencil [implicit]
- DACreate( )
- DA AO
- DAGlobalToLocal( )

**unstructured mesh**

- elements
- edges
- vertices
- VecScatterCreate( )
- VecScatter AO
- VecScatter( )

**Data Layout**

- beginner
- intermediate
Interfacing to 3\textsuperscript{rd} 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**

```
#include <tao.h>

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(…)
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)

- 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)
Three Ways of Using PETSc

SNESCreate() → SNES/KSP → MatCreate() → DAGetMat() → DM/DA → DAGetGlobalVec() → Manage yourself → VecCreate() → DMMGSetSNESLocal() → Grid Info → DMMG → vectors matrices → Solvers → DMMG
DMMG  

- Creates the vectors, matrices, ... appropriate for your grid.
- Connects them to a (hierarchy of) solvers
- User provides “local physics”
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