The **P**ortable **E**xtensible **T**oolkit for **S**cientific **C**omputing

Toby Isaac (building on slides from Jed Brown and Matt Knepley)

PETSc Tutorial
PETSc User Meeting
Atlanta, GA June 5, 2019

Welcome!

Thank you all for coming to our meeting!

- `https://www.mcs.anl.gov/petsc/meetings/2019/` for all updates
- Attendees from US, Italy, Germany, Saudi Arabia, Australia, Switzerland, & UK
- **wifi:** `eduroam` login credentials on your badges

Thanks!

- **The Institute for Data Engineering and Science** (`ideas.gatech.edu`) for travel support for student attendees
  - **Students:** save receipts, we will have a folder for you; reimbursement will come by check.
- **Fluid Numerics** (`fluidnumerics.com`) for support with Google Cloud (more in a minute. . .)
- **Anna Stroup-Holladay** for administrative support and planning

Today

- **Me**: Introduction, PETSc basics and data structures
- **Lunch** (on site)
- **Richard Mills**: Performance measurement & analysis, CPUs vs. GPUs
- **Jed Brown**: Time integrators
- **Matt Knepley**: Data management & meshes, nonlinear solvers

Tutorials today are hands-on

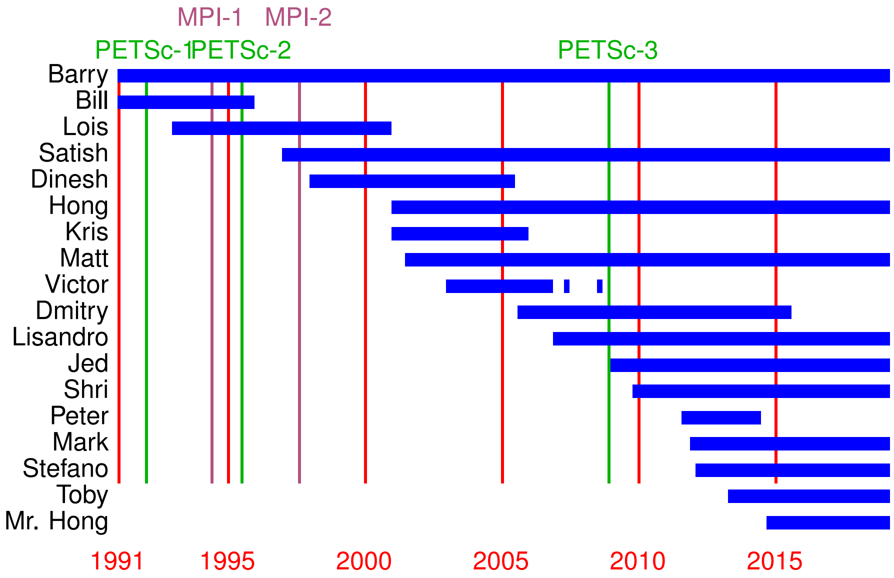
- Instructions: `https://bit.ly/2JWiWz9` (today only)
  - When opening Google Cloud Shell, may have to select 'fluidnumerics-cluster' from + dropdown
  - I had to use a vanilla browser (script/ad-blockers interfere)
- Slack messages in will show up here (intentionally, for once!): `fluidnumerics.slack.com`, channel `#petsc19`

Pause for Exercise 0: Getting logged in

Outline

Follow Up; Getting Help

- `http://www.mcs.anl.gov/petsc`
- Public questions: `petsc-users@mcs.anl.gov`, archived
- Private questions: `petsc-maint@mcs.anl.gov`, not archived


MPI-1
MPI-2
PETSc-1
PETSc-2
PETSc-3
Barry
Bill
Lois
Satish
Dinesh
Hong
Kris
Matt
Victor
Dmitry
Lisandro
Jed
Shri
Peter
Mark
Stefano
Toby
Mr. Hong
1991
1995
2000
2005
2010
2015

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. Cray, Blue Gene)
  - loosely coupled such as network of workstations
  - GPU clusters (many vector and sparse matrix kernels)
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double/quad precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Free to everyone (2-clause BSD license), open development
- 10^{12} unknowns, full-machine scalability on Top-10 systems
- Same code runs performantly on a laptop
- ~~No~~ iPhone support

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. Cray, Blue Gene)
  - loosely coupled such as network of workstations
  - GPU clusters (many vector and sparse matrix kernels)
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double/quad precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Free to everyone (2-clause BSD license), open development
- 10^{12} unknowns, full-machine scalability on Top-10 systems
- Same code runs performantly on a laptop
- ~~No~~ iPhone support

Portable **Extensible** Toolkit for Scientific computing

Philosophy: Everything has a plugin architecture

- Vectors, Matrices, Coloring/ordering/partitioning algorithms
- Preconditioners, Krylov accelerators
- Nonlinear solvers, Time integrators
- Spatial discretizations/topology*

Example

Vendor supplies matrix format and associated preconditioner, distributes compiled shared library. Application user loads plugin at runtime, no source code in sight.

Portable Extensible **Toolkit** for Scientific computing

Algorithms, (parallel) debugging aids, low-overhead profiling

Composability

Try new algorithms by choosing from product space and composing existing algorithms (multilevel, domain decomposition, splitting).

Experimentation

- It is not possible to pick the solver a priori.
  What will deliver best/competitive performance for a given physics, discretization, architecture, and problem size?
- PETSc's response: expose an algebra of composition so new solvers can be created at runtime.
- Important to keep solvers decoupled from physics and discretization because we also experiment with those.

Portable Extensible Toolkit for **Scientific computing**

- Computational Scientists
  - PyLith (CIG), Underworld (Monash), Climate (ICL/UK Met), PFLOTRAN (DOE), MOOSE (DOE), Proteus (ERDC)
- Algorithm Developers (iterative methods and preconditioning)
- Package Developers
  - SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM
- Funding
  - Department of Energy
    - SciDAC, ASCR ISICLES, MICS Program, INL Reactor Program
  - National Science Foundation
    - CIG, CISE, Multidisciplinary Challenge Program
- Hundreds of tutorial-style examples
- Hyperlinked manual, examples, and manual pages for all routines
- Support from `petsc-maint@mcs.anl.gov`

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
    - UNIC on BG/P and XT5
    - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
    - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)
    - Jed Brown on NERSC Edison
    - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
  - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
  - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)
  - Jed Brown on NERSC Edison
  - HPGMG code

TECH MICROSOFT MICROSOFT BUILD

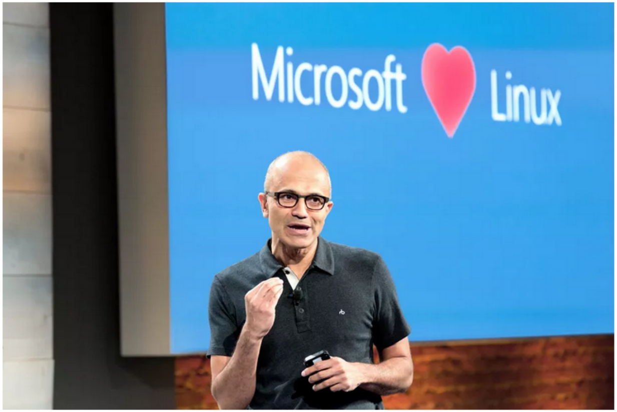
Microsoft will ship a full Linux kernel in Windows 10

74

Available in testing this summer

By Tom Warren | @tomwarren | May 6, 2019, 7:24pm EDT

SHARE



GOOD DEALS

New Fortnite edition purple Xbox One S will go on sale on June 7th

Microsoft has surprised many in the Linux developer community in recent years. Surprises

Outline

Downloading PETSc

- The latest tarball is on the PETSc site: http://www.mcs.anl.gov/petsc/download
- There is a Debian package (`aptitude install petsc-dev`)
- There is a Git development repository

Cloning PETSc

- The full development repository is open to the public
  - https://bitbucket.org/petsc/petsc/
- Why is this better?
  - You can clone to any release (or any specific ChangeSet)
  - You can easily rollback changes (or releases)
  - You can get fixes from us the same day
  - You can easily submit changes using a pull request
- All releases are just tags:
  - Source at tag v3.10.3

Unpacking PETSc

- Just clone development repository
  - `git clone http://bitbucket.org/petsc/petsc.git`
  - `git checkout -rv3.10.3`

or

- Unpack the tarball
  - `tar xzf petsc.tar.gz`

Outline

Configuring PETSc

- Set `$PETSC_DIR` to the installation root directory
- Run the configuration utility
  - `$PETSC_DIR/configure`
  - `$PETSC_DIR/configure --help`
  - `$PETSC_DIR/configure --download-mpich`
  - `$PETSC_DIR/configure --prefix=/usr`
- There are many examples in `$PETSC_DIR/config/examples`
- Config files in `$PETSC_DIR/$PETSC_ARCH/lib/petsc/conf`
  - Config header in `$PETSC_DIR/$PETSC_ARCH/include`
  - `$PETSC_ARCH` has a default if not specified

Configuring PETSc

- You can easily reconfigure with the same options
  - `./$PETSC_ARCH/lib/petsc/conf/reconfigure-$PETSC_ARCH.py`
- Can maintain several different configurations
  - `./configure -PETSC_ARCH=arch-linux-opt --with-debugging=0`
- All configuration information is in the logfile
  - `./$PETSC_ARCH/lib/petsc/conf/configure.log`
  - ALWAYS send this file with bug reports

Automatic Downloads

- Starting in 2.2.1, some packages are automatically
  - Downloaded
  - Configured and Built (in `$PETSC_DIR/externalpackages`)
  - Installed with PETSc
- Currently works for
  - petsc4py, mpi4py
  - PETSc documentation utilities (Sowing, c2html)
  - BLAS, LAPACK, Elemental, ScaLAPACK
  - MPICH, OpenMPI
  - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
  - SuiteSparse, MUMPS, SuperLU, SuperLU_Dist, PaStiX, Pardiso
  - HYPRE, ML
  - BLOPEX, FFTW, STRUMPACK, SPAI, CUSP, Sundials
  - Triangle, TetGen, p4est, Pragmatic
  - HDF5, NetCDF, ExodusII
  - AfterImage, gifLib, libjpeg, opengl
  - GMP, MPFR
  - ConcurrencyKit, hwloc

Outline

Building PETSc

- There is now One True Way to build PETSc:
  - `make`
  - `make install` if you configured with `--prefix`
  - Check build when done with `make check`
- Can build multiple configurations
  - `PETSC_ARCH=arch-linux-opt make`
  - Libraries are in `$PETSC_DIR/$PETSC_ARCH/lib/`
- Complete log for each build is in logfile
  - `./$PETSC_ARCH/lib/petsc/conf/make.log`
  - ALWAYS send this with bug reports

Outline

Running PETSc

- Try running PETSc examples first
  - `cd $PETSC_DIR/src/snes/examples/tutorials`
- Build examples using make targets
  - `make ex5`
- Run examples using the make target
  - `make runex5`
- Can also run using MPI directly
  - `mpirun ./ex5 -snes_max_it 5`
  - `mpiexec ./ex5 -snes_monitor`

Exercise 1

Run SNES Example 5 using come custom options.

1. `cd $PETSC_DIR/src/snes/examples/tutorials`
2. `make ex5`
3. `mpiexec ./ex5 -snes_monitor -snes_view`
4. `mpiexec ./ex5 -snes_type tr -snes_monitor -snes_view`
5. `mpiexec ./ex5 -ksp_monitor -snes_monitor -snes_view`
6. `mpiexec ./ex5 -pc_type jacobi -ksp_monitor -snes_monitor -snes_view`
7. `mpiexec ./ex5 -ksp_type bicg -ksp_monitor -snes_monitor -snes_view`

Running PETSc

- PETSc has a new test infrastructure
  - Described in Manual Section 1.3 and the Developer's Guide
- Run all tests
  - `make PETSC_ARCH=arch-myarch test`
- Run a specific example
  - `make -f gmakefile test search='vec_vec_tutorials-ex6'`
- Run a set of similar examples
  - `make -f gmakefile test globsearch='ts*'`
  - `make -f gmakefile test globsearch='ts_tutorials-ex11_*'`
  - `make -f gmakefile test argsearch='cuda'`

Using MPI

- The Message Passing Interface is:
  - a library for parallel communication
  - a system for launching parallel jobs (mpirun/mpiexec)
  - a community standard
- Launching jobs is easy
  - `mpiexec -n 4 ./ex5`
- You should never have to make MPI calls when using PETSc
  - Almost never

MPI Concepts

- Communicator
  - A context (or scope) for parallel communication (“Who can I talk to”)
  - There are two defaults:
    - yourself (PETSC_COMM_SELF),
    - and everyone launched (PETSC_COMM_WORLD)
  - Can create new communicators by splitting existing ones
  - Every PETSc object has a communicator
  - Set PETSC_COMM_WORLD to put all of PETSc in a subcomm
- Point-to-point communication
  - Happens between two processes (like in MatMult())
- Reduction or scan operations
  - Happens among all processes (like in VecDot())

Common Viewing Options

- Gives a text representation
  - `-vec_view`
- Generally views subobjects too
  - `-snes_view`
- Can visualize some objects
  - `-mat_view draw::`
- Alternative formats
  - `-vec_view binary:sol.bin:`, `-vec_view ::matlab`, `-vec_view socket`
- Sometimes provides extra information
  - `-mat_view ::ascii_info`, `-mat_view ::ascii_info_detailed`
- Use **`-help`** to see all options

Common Monitoring Options

- Display the residual
  - `-ksp_monitor`, graphically `-ksp_monitor_draw`
- Can disable dynamically
  - `-ksp_monitors_cancel`
- Does not display subsolvers
  - `-snes_monitor`
- Can use the true residual
  - `-ksp_monitor_true_residual`
- Can display different subobjects
  - `-snes_monitor_residual`, `-snes_monitor_solution`, `-snes_monitor_solution_update`
  - `-snes_monitor_range`
  - `-ksp_gmres_krylov_monitor`
- Can display the spectrum
  - `-ksp_monitor_singular_value`

Outline

Getting More Help

- http://www.mcs.anl.gov/petsc
- Hyperlinked documentation
  - Manual
  - Manual pages for every method
  - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov

Outline

- C

subsectionOur motivating example for today

A straight-line code

`https://bitbucket.org/tisaac/petsc19-tutorial-morning-demo`

Solves $I + vv^T = b$.

How do we take this straight-line code to one that exploits configuration, extensibility, and other PETSc design patterns?

Outline

Application Integration

- Be willing to experiment with algorithms
  - No optimality without interplay between physics and algorithmics
- Adopt flexible, extensible programming
  - Algorithms and data structures not hardwired
- Be willing to play with the real code
  - Toy models are rarely helpful
- If possible, profile before integration
  - Automatic in PETSc

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
  - Python

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

Integration Stages

- Version Control
    - It is impossible to overemphasize
    - We use Git
- Initialization
    - Linking to PETSc
- Profiling
    - Profile before changing
    - Also incorporate command line processing
- Linear Algebra
    - First PETSc data structures
- Solvers
    - Very easy after linear algebra is integrated

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

- Use `-log_view` for a performance profile
  - Event timing
  - Event flops
  - Memory usage
  - MPI messages

  This used to be `-log_summary`
- Call PetscLogStagePush() and PetscLogStagePop()
  - User can add new stages
- Call PetscLogEventBegin() and PetscLogEventEnd()
  - User can add new events

Command Line Processing

- Check for an option
  - PetscOptionsHasName()
- Retrieve a value
  - PetscOptionsGetInt(), PetscOptionsGetIntArray()
- Set a value
  - PetscOptionsSetValue()
- Check for unused options
  - `-options_left`
- Clear, alias, reject, etc.
- Modern form uses
  - PetscOptionsBegin(), PetscOptionsEnd()
  - PetscOptionsInt(), PetscOptionsReal()
  - Integrates with **`-help`**

Outline

Vector Algebra

What are PETSc vectors?

- Fundamental objects representing
    - solutions
    - right-hand sides
    - coefficients
- Each process locally owns a subvector of contiguous global data

Vector Algebra

How do I create vectors?

- VecCreate(MPI_Commcomm, Vec*v)
- VecSetSizes(Vecv, PetscInt n, PetscInt N)
- VecSetType(Vecv, VecType typeName)
- VecSetFromOptions(Vecv)
    - Can set the type at runtime

Vector Algebra

A PETSc Vec

- Supports all vector space operations
  - VecDot(), VecNorm(), VecScale()
- Has a direct interface to the values
  - VecGetArray(), VecGetArrayF90()
- Has unusual operations
  - VecSqrtAbs(), VecStrideGather()
- Communicates automatically during assembly
- Has customizable communication (PetscSF, VecScatter)

Parallel Assembly
Vectors and Matrices

- Processes may set an arbitrary entry
  - Must use proper interface
- Entries need not be generated locally
  - Local meaning the process on which they are stored
- PETSc automatically moves data if necessary
  - Happens during the assembly phase

Vector Assembly

- A three step process
  - Each process sets or adds values
  - Begin communication to send values to the correct process
  - Complete the communication

```
VecSetValues(Vec v, PetscInt n, PetscInt rows[],
             PetscScalar values[], InsertMode mode)
```

- Mode is either INSERT_VALUES or ADD_VALUES
- Two phases allow overlap of communication and computation
  - VecAssemblyBegin(v)
  - VecAssemblyEnd(v)

One Way to Set the Elements of a Vector

```
ierr = VecGetSize(x, &N);CHKERRQ(ierr);
ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank);CHKERRQ(ierr);
if (rank == 0) {
  val = 0.0;
  for(i = 0; i < N; ++i) {
    ierr = VecSetValues(x, 1, &i, &val, INSERT_VALUES);CHKERRQ(ierr);
    val += 10.0;
  }
}
/* These routines ensure that the data is
   distributed to the other processes */
ierr = VecAssemblyBegin(x);CHKERRQ(ierr);
ierr = VecAssemblyEnd(x);CHKERRQ(ierr);
```

One Way to Set the Elements of a Vector

```
VecGetSize(x, &N);
MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
if (rank == 0) {
  val = 0.0;
  for(i = 0; i < N; ++i) {
    VecSetValues(x, 1, &i, &val, INSERT_VALUES);
    val += 10.0;
  }
}
/* These routines ensure that the data is
   distributed to the other processes */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

A Better Way to Set the Elements of a Vector

```
VecGetOwnershipRange(x, &low, &high);
val = low*10.0;
for(i = low; i < high; ++i) {
  VecSetValues(x, 1, &i, &val, INSERT_VALUES);
  val += 10.0;
}
/* No data will be communicated here */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

Selected Vector Operations

Function Name	Operation
VecAXPY(Vec y, PetscScalar a, Vec x)	$y = y + a * x$
VecAYPX(Vec y, PetscScalar a, Vec x)	$y = x + a * y$
VecWAYPX(Vec w, PetscScalar a, Vec x, Vec y)	$w = y + a * x$
VecScale(Vec x, PetscScalar a)	$x = a * x$
VecCopy(Vec y, Vec x)	$y = x$
VecPointwiseMult(Vec w, Vec x, Vec y)	$w_i = x_i * y_i$
VecMax(Vec x, PetscInt *idx, PetscScalar *r)	$r = \max r_i$
VecShift(Vec x, PetscScalar r)	$x_i = x_i + r$
VecAbs(Vec x)	$x_i = \|x_i\|$
VecNorm(Vec x, NormType type, PetscReal *r)	$r = \|\|x\|\|$

Working With Local Vectors

It is sometimes more efficient to directly access local storage of a `Vec`.

- 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
  - Commonly, these routines are fast and do not involve a copy

VecGetArray in C

```
Vec            v;
PetscScalar   *array;
PetscInt       n, i;

VecGetArray(v, &array);
VecGetLocalSize(v, &n);
PetscSynchronizedPrintf(PETSC_COMM_WORLD,
  "First element of local array is %f\n", array[0]);
PetscSynchronizedFlush(PETSC_COMM_WORLD);
for(i = 0; i < n; ++i) {
  array[i] += (PetscScalar) rank;
}
VecRestoreArray(v, &array);
```

VecGetArray in F77

```
#include "finclude/petsc.h"

      Vec             v;
      PetscScalar     array(1)
      PetscOffset     offset
      PetscInt        n, i
      PetscErrorCode  ierr

      call VecGetArray(v, array, offset, ierr)
      call VecGetLocalSize(v, n, ierr)
      do i=1,n
        array(i+offset) = array(i+offset) + rank
      end do
      call VecRestoreArray(v, array, offset, ierr)
```

VecGetArray in F90

```
#include "finclude/petsc.h90"

      Vec             v;
      PetscScalar     pointer :: array(:)
      PetscInt        n, i
      PetscErrorCode  ierr


      call VecGetArrayF90(v, array, ierr)
      call VecGetLocalSize(v, n, ierr)
      do i=1,n
        array(i) = array(i) + rank
      end do
      call VecRestoreArrayF90(v, array, ierr)
```

VecGetArray in Python

```
with v as a:
  for i in range(len(a)):
    a[i] = 5.0*i
```

DMDAVecGetArray in C

```
DM                da;
Vec               v;
DMDALocalInfo  *info;
PetscScalar    **array;

DMDAVecGetArray(da, v, &array);
for(j = info->ys; j < info->ys+info->ym; ++j) {
  for(i = info->xs; i < info->xs+info->xm; ++i) {
    u       = x[j][i];
    uxx     = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
    uyy     = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
    f[j][i] = uxx + uyy;
  }
}
DMDAVecRestoreArray(da, v, &array);
```

DMDAVecGetArray in F90

```
DM                  da
Vec                 v
PetscScalar,pointer :: array(:,:)

call DMDAGetCorners(ada,xs,ys,PETSC_NULL_INTEGER,
                          xm,ym,PETSC_NULL_INTEGER,ierr)
call DMDAVecGetArrayF90(da,v,array,ierr);
do i = xs,xs+xm
  do j = ys,ys+ym
    u      = x(i,j)
    uxx    = (2.0*u - x(i-1,j) - x(i+1,j))*hydhx;
    uyy    = (2.0*u - x(i,j-1) - x(i,j+1)*hxdhy;
    f(i,j) = uxx + uyy;
  enddo
enddo
call DMDAVecRestoreArrayF90(da,v,array,ierr);
```

Outline

Matrix Algebra

What are PETSc matrices?

- Fundamental objects for storing stiffness matrices and Jacobians
- Each process locally owns a contiguous set of rows
- Supports many data types
  - AIJ, Block AIJ, Symmetric AIJ, Block Matrix, etc.
- Supports structures for many packages
  - Elemental, MUMPS, SuperLU, UMFPack, PasTiX

How do I create matrices?

- MatCreate(MPI_Comm comm, Mat *A)
- MatSetSizes(Mat A, PetscInt m, PetscInt n, PetscInt M, PetscInt N)
- MatSetType(Mat A, MatType typeName)
- MatSetFromOptions(Mat A)
  - Can set the type at runtime
- MatSeqAIJPreallocation(Mat A, PetscInt nz, const PetscInt nnz[])
- MatXAIJPreallocation(Mat A, bs, dnz[], onz[], dnzu[], onzu[])
- MatSetValues(Mat A, m, rows[], n, cols [], values [], InsertMode)
  - **MUST** be used, but does automatic communication

Matrix Polymorphism

The PETSc Mat has a single user interface,

- Matrix assembly
  - MatSetValues()
  - MatGetLocalSubMatrix()
- Matrix-vector multiplication
  - MatMult()
- Matrix viewing
  - MatView()

but multiple underlying implementations.

- AIJ, Block AIJ, Symmetric Block AIJ,
- Dense
- Matrix-Free
- etc.

A matrix is defined by its interface, not by its data structure.

Matrix Assembly

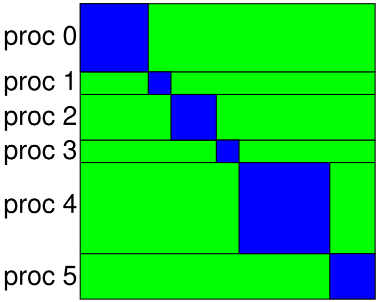
- A three step process
  - Each process sets or adds values
  - Begin communication to send values to the correct process
  - Complete the communication
- MatSetValues(A, m, rows[], n, cols [], values [], mode)
  - mode is either INSERT_VALUES or ADD_VALUES
  - Logically dense block of values
- Two phase assembly allows overlap of communication and computation
  - MatAssemblyBegin(A, type)
  - MatAssemblyEnd(A, type)
  - type is either MAT_FLUSH_ASSEMBLY or MAT_FINAL_ASSEMBLY

One Way to Set the Elements of a Matrix

Simple 3-point stencil for 1D Laplacian

```
v[0] = -1.0; v[1] = 2.0; v[2] = -1.0;
if (rank == 0) {
  for(row = 0;  row < N; row++) {
    cols[0] = row-1; cols[1] = row; cols[2] = row+1;
    if (row == 0) {
      MatSetValues(A,1,&row,2,&cols[1],&v[1],INSERT_VALUES);
    } else if (row == N-1) {
      MatSetValues(A,1,&row,2,cols,v,INSERT_VALUES);
    } else {
      MatSetValues(A,1,&row,3,cols,v,INSERT_VALUES);
    }
  }
}
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

Parallel Sparse Matrix Layout



🟦 diagonal blocks
🟩 offdiagonal blocks

A Better Way to Set the Elements of a Matrix

Simple 3-point stencil for 1D Laplacian

```
v[0] = -1.0; v[1] = 2.0; v[2] = -1.0;
MatGetOwnershipRange(A,&start,&end);
for(row = start; row < end; row++) {
  cols[0] = row-1; cols[1] = row; cols[2] = row+1;
  if (row == 0) {
    MatSetValues(A,1,&row,2,&cols[1],&v[1],INSERT_VALUES);
  } else if (row == N-1) {
    MatSetValues(A,1,&row,2,cols,v,INSERT_VALUES);
  } else {
    MatSetValues(A,1,&row,3,cols,v,INSERT_VALUES);
  }
}
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

Why Are PETSc Matrices That Way?

- No one data structure is appropriate for all problems
  - Blocked and diagonal formats provide performance benefits
  - PETSc has many formats
  - Makes it easy to add new data structures
- Assembly is difficult enough without worrying about partitioning
  - PETSc provides parallel assembly routines
  - High performance still requires making most operations local
  - However, programs can be incrementally developed.
  - MatPartitioning and MatOrdering can help
  - Its better to partition and reorder the underlying grid
- Matrix decomposition in contiguous chunks is simple
  - Makes interoperation with other codes easier
  - For other ordering, PETSc provides “Application Orderings” (AO)

Outline

Experimentation is Essential!

Proof is not currently 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**, pp.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.

Linear Solvers
Krylov Methods

- Using PETSc linear algebra, just add:
  - KSPSetOperators(ksp, A, M, flag)
  - KSPSolve(ksp, b, x)
- Can access subobjects
  - KSPGetPC(ksp, &pc)
- Preconditioners must obey PETSc interface
  - Basically just the KSP interface
- Can change solver dynamically from the command line
  - `-ksp_type bicgstab`

Nonlinear Solvers

- Using PETSc linear algebra, just add:
  - SNESSetFunction(snes, r, residualFunc, ctx)
  - SNESSetJacobian(snes, A, M, jacFunc, ctx)
  - SNESSolve(snes, b, x)
- Can access subobjects
  - SNESGetKSP(snes, &ksp)
- Can customize subobjects from the cmd line
  - Set the subdomain preconditioner to ILU with −sub_pc_type ilu

Basic Solver Usage

Use SNESSetFromOptions() so that everything is set dynamically

- Set the type
  - Use −snes_type (or take the default)
- Set the preconditioner
  - Use −npc_snes_type (or take the default)
- Override the tolerances
  - Use −snes_rtol and −snes_atol
- View the solver to make sure you have the one you expect
  - Use −snes_view
- For debugging, monitor the residual decrease
  - Use −snes_monitor
  - Use −ksp_monitor to see the underlying linear solver

3rd Party Solvers in PETSc

Complete table of solvers

- Sequential LU
    - ESSL (IBM)
    - SuperLU (Sherry Li, LBNL)
    - Suitesparse (Tim Davis, U. of Florida)
    - LUSOL (MINOS, Michael Saunders, Stanford)
    - PILUT (Hypre, David Hysom, LLNL)
- Parallel LU
    - Elemental/Clique (Jack Poulson, Google)
    - MUMPS (Patrick Amestoy, IRIT)
    - SuperLU_Dist (Jim Demmel and Sherry Li, LBNL)
    - Pardiso (MKL, Intel)
    - STRUMPACK (Pieter Ghysels, LBNL)
- Parallel Cholesky
    - Elemental (Jack Poulson, Google)
    - DSCPACK (Padma Raghavan, Penn. State)
    - MUMPS (Patrick Amestoy, Toulouse)

3rd Party Preconditioners in PETSc

Complete table of solvers

- Parallel Algebraic Multigrid
  - GAMG (Mark Adams, LBNL)
  - BoomerAMG (Hypre, LLNL)
  - ML (Trilinos, Ray Tuminaro and Jonathan Hu, SNL)
- Parallel BDDC (Stefano Zampini, KAUST)
- Parallel ILU, PaStiX (Faverge Mathieu, INRIA)
- Parallel Redistribution (Dave May, Oxford and Patrick Sanan, USI)
- Parallel Sparse Approximate Inverse
  - Parasails (Hypre, Edmund Chow, LLNL)
  - SPAI 3.0 (Marcus Grote and Barnard, NYU)

User Solve

```
MPI_Comm comm;
SNES snes;
DM dm;
Vec u;

SNESCreate(comm, &snes);
SNESSetDM(snes, dm);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, NULL, u);
```

Solver use in SNES ex62

Solver code does not change for different algorithms:

```
SNES            snes;
DM              dm;
Vec             u;
PetscErrorCode  ierr;

ierr = SNESCreate(PETSC_COMM_WORLD, &snes);CHKERRQ(ierr);
ierr = SNESSetDM(snes, dm);CHKERRQ(ierr);
/* Specify residual computation */
ierr = SNESSetFromOptions(snes);CHKERRQ(ierr); /* Configure solver */
ierr = DMCreateGlobalVector(dm, &u);CHKERRQ(ierr);
ierr = SNESSolve(snes, PETSC_NULL, u);CHKERRQ(ierr);
```

- Never recompile! all configuration is dynamic
- DM controls data layout and communication
- Type of nested solvers can be changed at runtime

Solver use in SNES ex62

I will omit error checking and declarations:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
/* Specify residual computation */
SNESSetFromOptions(snes); /* Configure solver */
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);
```

Solver use in SNES ex62

The configuration API can also be used:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
/* Specify residual computation */
SNESNGMRESSetRestartType(snes, SNES_NGMRES_RESTART_PERIODIC);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);
```

- Ignored when not applicable (no ugly check)
- Type safety of arguments is retained
- No downcasting

Solver use in SNES ex62

Adding a prefix namespaces command line options:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
/* Specify residual computation */
SNESSetOptionsPrefix(snes, "stokes_");
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);
```

`-stokes_snes_type qn` changes the solver type,

whereas `-snes_type qn` does not

Solver use in SNES ex62

User provides a function to compute the residual:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
DMCreateGlobalVector(dm, &r);
SNESSetFunction(snes, r, FormFunction, &user);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);
```

$$r = F(u)$$

- User handles parallel communication
- User handles domain geometry and discretization

Solver use in SNES ex62

DM allows the user to compute only on a local patch:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);

DMSNESSetLocalFunction(dm, FormFunctionLocal);
```

- Code looks serial to the user
- PETSc handles global residual assembly
- Also works for unstructured meshes

Solver use in SNES ex62

Optionally, the user can also provide a Jacobian:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);

DMSNESSetLocalFunction(dm, FormFunctionLocal);
DMSNESSetLocalJacobian(dm, FormJacobianLocal);
```

SNES ex62 allows both

- finite difference (JFNK), and
- FEM action

versions of the Jacobian.

Solver use in SNES ex62

Convenience form uses Plex defaults:

```
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, dm);
SNESSetFromOptions(snes);
DMCreateGlobalVector(dm, &u);
SNESSolve(snes, PETSC_NULL, u);

DMPlexSetSNESLocalFEM(dm,&user,&user,&user);
```

This also handles Dirichlet boundary conditions.

Solver use in SNES ex62

The DM also handles storage:

```
 CreateMesh(PETSC_COMM_WORLD, &user, &dm);
DMCreateLocalVector(dm, &lu);
DMCreateGlobalVector(dm, &u);
DMCreateMatrix(dm, &J);
```

- DM can create local and global vectors
- Matrices are correctly preallocated
- Easy supported for discretization

Outline

Correctness Debugging

- Automatic generation of tracebacks
- Detecting memory corruption and leaks
- Optional user-defined error handlers

Interacting with the Debugger

- Launch the debugger
  - `-start_in_debugger [gdb,dbx,noxterm]`
  - `-on_error_attach_debugger [gdb,dbx,noxterm]`
- Attach the debugger only to some parallel processes
  - `-debugger_nodes 0,1`
- Set the display (often necessary on a cluster)
  - `-display khan.mcs.anl.gov:0.0`

Debugging Tips

- Put a breakpoint in PetscError() to catch errors as they occur
- PETSc tracks memory overwrites at both ends of arrays
  - The CHKMEMQ macro causes a check of all allocated memory
  - Track memory overwrites by bracketing them with CHKMEMQ
- PETSc checks for leaked memory
  - Use PetscMalloc() and PetscFree() for all allocation
  - Print unfreed memory on PetscFinalize() with `-malloc_dump`
- Simply the best tool today is valgrind
  - It checks memory access, cache performance, memory usage, etc.
  - http://www.valgrind.org
  - Need `--trace-children=yes` when running under MPI

Outline

Performance Debugging

- PETSc has integrated profiling
  - Option `-log_view` prints a report on PetscFinalize()
- PETSc allows user-defined events
  - Events report time, calls, flops, communication, etc.
  - Memory usage is tracked by object
- Profiling is separated into stages
  - Event statistics are aggregated by stage

Using Stages and Events

- Use PetscLogStageRegister() to create a new stage
  - Stages are identifier by an integer handle
- Use PetscLogStagePush/Pop() to manage stages
  - Stages may be nested, but will not aggregate in a nested fashion
- Use PetscLogEventRegister() to create a new stage
  - Events also have an associated class
- Use PetscLogEventBegin/End() to manage events
  - Events may also be nested and will aggregate in a nested fashion
  - Can use PetscLogFlops() to log user flops

Adding A Logging Stage
C

```
int stageNum;

PetscLogStageRegister(&stageNum, "name");
PetscLogStagePush(stageNum);

/* Code to Monitor */

PetscLogStagePop();
```

Adding A Logging Stage
Python

```
with PETSc.LogStage('Fluid Stage') as fluidStage:
  # All operations will be aggregated in fluidStage
  fluid.solve()
```

Adding A Logging Event
C

```
static int USER_EVENT;

PetscLogEventRegister(&USER_EVENT, "name", CLS_ID);
PetscLogEventBegin(USER_EVENT,0,0,0,0);

/* Code to Monitor */

PetscLogFlops(user_event_flops);
PetscLogEventEnd(USER_EVENT,0,0,0,0);
```

Adding A Logging Event
Python

```
with PETSc.logEvent('Reconstruction') as recEvent:
  # All operations are timed in recEvent
  reconstruct(sol)
  # Flops are logged to recEvent
  PETSc.Log.logFlops(user_event_flops)
```

Adding A Logging Class

```
static int CLASS_ID;
PetscLogClassRegister(&CLASS_ID, "name");
```

- Class ID identifies a class uniquely
- Must initialize before creating any objects of this type

Matrix Memory Preallocation

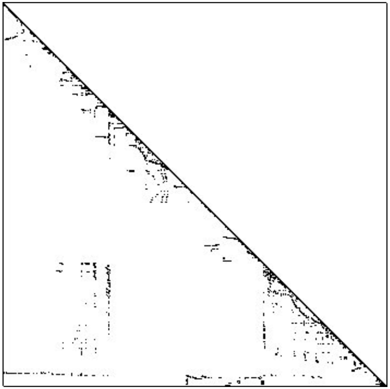
- PETSc sparse matrices are dynamic data structures
  - can add additional nonzeros freely
- Dynamically adding many nonzeros
  - requires additional memory allocations
  - requires copies
  - can kill performance
- Memory preallocation provides
  - the freedom of dynamic data structures
  - good performance
- Easiest solution is to replicate the assembly code
  - Remove computation, but preserve the indexing code
  - Store set of columns for each row
- Call preallocation rourines for all datatypes
  - MatSeqAIJSetPreallocation()
  - MatMPIAIJSetPreallocation()
  - Only the relevant data will be used

Matrix Memory Preallocation
Sequential Sparse Matrices

```
MatSeqAIJPreallocation(Mat A, int nz, int nnz[])
```

nz: expected number of nonzeros in any row

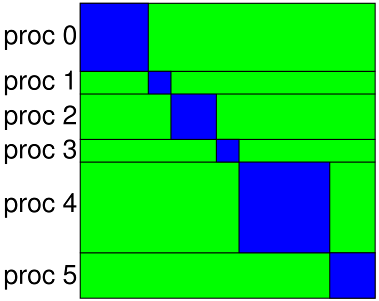
nnz(i): expected number of nonzeros in row i



Matrix Memory Preallocation

ParallelSparseMatrix

- Each process locally owns a submatrix of contiguous global rows
- Each submatrix consists of diagonal and off-diagonal parts



🟦 diagonal blocks

🟩 offdiagonal blocks

- MatGetOwnershipRange(MatA,int *start,int *end)

`start`: first locally owned row of global matrix

`end-1`: last locally owned row of global matrix

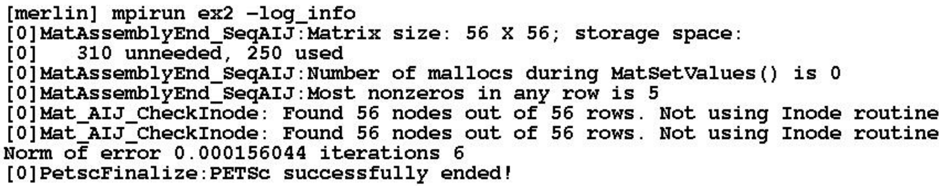
Matrix Memory Preallocation
Parallel Sparse Matrices

```
MatMPIAIJPreallocation(Mat A, int dnz, int dnnz[], int onz, int onnz[])
```

dnz:	expected number of nonzeros in any row in the diagonal block
nnz(i):	expected number of nonzeros in row i in the diagonal block
onz:	expected number of nonzeros in any row in the offdiagonal portion
nnz(i):	expected number of nonzeros in row i in the offdiagonal portion

Matrix Memory Preallocation
Verifying Preallocation

- Use runtime option `-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 routine
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routine
Norm of error 0.000156044 iterations 6
[0]PetscFinalize:PETSc successfully ended!
```

Outline

DM Interface

- Allocation
  - `DMCreateGlobalVector(DM, Vec *)`
  - `DMCreateLocalVector(DM, Vec *)`
  - `DMCreateMatrix(DM, MatType, Mat *)`
- Mapping
  - `DMGlobalToLocalBegin/End(DM, Vec, InsertMode, Vec)`
  - `DMLocalToGlobalBegin/End(DM, Vec, InsertMode, Vec)`
  - `DMGetLocalToGlobalMapping(DM, IS *)`

DM Interface

- Geometry
  - `DMGetCoordinateDM(DM, DM *)`
  - `DMGetCoordinates(DM, Vec *)`
  - `DMGetCoordinatesLocal(DM, Vec *)`
- Layout
  - `DMGetDefaultSection(DM, PetscSection *)`
  - `DMGetDefaultGlobalSection(DM, PetscSection *)`
  - `DMGetDefaultSF(DM, PetscSF *)`

DM Interface

- Hierarchy
  - `DMRefine(DM, MPI_Comm, DM *)`
  - `DMCoarsen(DM, MPI_Comm, DM *)`
  - `DMGetSubDM(DM, MPI_Comm, DM *)`
- Intergrid transfer
  - `DMGetInterpolation(DM, DM, Mat *, Vec *)`
  - `DMGetAggregates(DM, DM, Mat *)`
  - `DMGetInjection(DM, DM, VecScatter *)`

Multigrid Paradigm

The **DM** interface uses the *local* callback functions to

- assemble global functions/operators from local pieces
- assemble functions/operators on coarse grids

Then **PCMG** organizes

- control flow for the multilevel solve, and
- projection and smoothing operators at each level.

Outline

What is a DMDA?

DMDA is a topology interface on structured grids

- Handles parallel data layout
- Handles local and global indices
    - `DMDAGetGlobalIndices()` and `DMDAGetAO()`
- Provides local and global vectors
    - `DMGetGlobalVector()` and `DMGetLocalVector()`
- Handles ghost values coherence
    - `DMGlobalToLocalBegin/End()` and `DMLocalToGlobalBegin/End()`

Residual Evaluation

The **DM** interface is based upon *local* callback functions

- `FormFunctionLocal()`
- `FormJacobianLocal()`

Callbacks are registered using

- `SNESSetDM()`, `TSSetDM()`
- `DMSNESSetFunctionLocal()`, `DMTSSetJacobianLocal()`

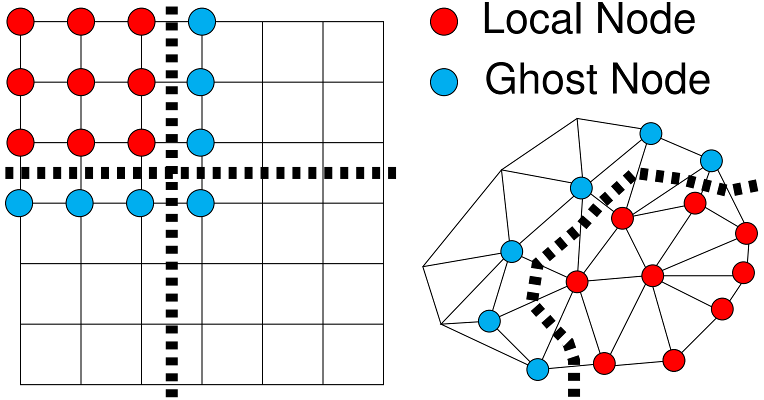
When PETSc needs to evaluate the nonlinear residual **F(x)**,

- Each process evaluates the local residual
- PETSc assembles the global residual automatically
  - Uses `DMLocalToGlobal()` method

Ghost Values

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

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



DMDA Global Numberings

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

DMDA Global vs. Local Numbering

- **Global**: Each vertex has a unique id belongs on a unique process
- **Local**: Numbering includes vertices from neighboring processes
  - These are called ghost vertices

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

DMDA Local Function

User provided function calculates the nonlinear residual (in 2D)

```
(* lf )(DMDALocalInfo *info, PetscScalar**x, PetscScalar **r, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution (a multidimensional array)

`r`: The residual

`ctx`: The user context passed to DMDASNESSetFunctionLocal()

The local DMDA function is activated by calling

```
DMDASNESSetFunctionLocal(dm, INSERT_VALUES, lfunc, &ctx)
```

Bratu Residual Evaluation

$$\Delta u + \lambda e^u = 0$$

```
ResLocal(DMDALocalInfo *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) {
    u = x[j][i];
    if (i==0 || j==0 || i == M || j == N) {
      f[j][i] = 2.0*(hydhx+hxdhy)*u; continue;
    }
    u_xx     = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
    u_yy     = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
    f[j][i] = u_xx + u_yy - hx*hy*lambda*exp(u);
}}}
```

$PETSC_DIR/src/snes/examples/tutorials/ex5.c

DMDA Local Jacobian

User provided function calculates the Jacobian (in 2D)

```
(* ljac )(DMDALocalInfo *info, PetscScalar**x, Mat J, void *ctx)
```

- `info`: All layout and numbering information
- `x`: The current solution
- `J`: The Jacobian
- `ctx`: The user context passed to DASetLocalJacobian()

The local DMDA function is activated by calling

```
DMDASNESSetJacobianLocal(dm, ljac, &ctx)
```

Bratu Jacobian Evaluation

```
JacLocal(DMDALocalInfo *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 == mx-1 || j == 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);
}}}}
```

$PETSC_DIR/src/snes/examples/tutorials/ex5.c

DMDA Vectors

- The **DMDA** object contains only layout (topology) information
    - All field data is contained in PETSc **Vecs**
- Global vectors are parallel
    - Each process stores a unique local portion
    - `DMCreateGlobalVector(DM da, Vec *gvec)`
- Local vectors are sequential (and usually temporary)
    - Each process stores its local portion plus ghost values
    - `DMCreateLocalVector(DM da, Vec *lvec)`
    - includes ghost and boundary values!

Updating Ghosts

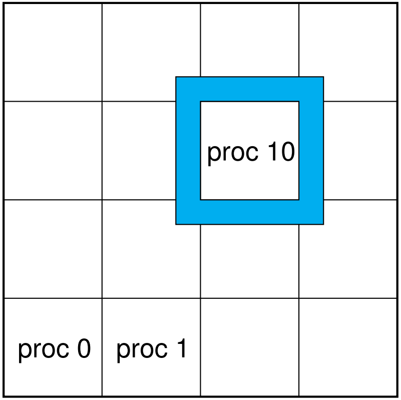
Two-step process enables overlapping computation and communication

- DMGlobalToLocalBegin(da, gvec, mode, lvec)
  - gvec provides the data
  - mode is either INSERT_VALUES or ADD_VALUES
  - lvec holds the local and ghost values
- DMGlobalToLocalEnd(da, gvec, mode, lvec)
  - Finishes the communication

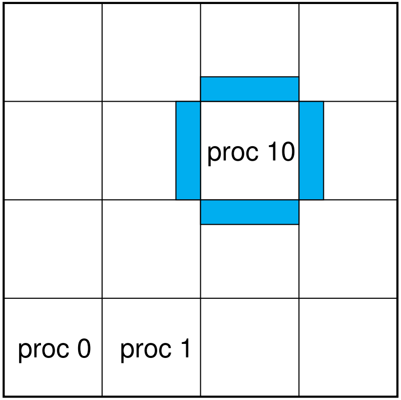
The process can be reversed with DALocalToGlobalBegin/End().

DMDA Stencils

Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

Setting Values on Regular Grids

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n, MatStencil idxn[],
                    PetscScalar values[], InsertMode mode)
```

- Each row or column is actually a **MatStencil**
  - This specifies grid coordinates and a component if necessary
  - Can imagine for unstructured grids, they are *vertices*
- The values are the same logically dense block in row/col

Creating a DMDA

DMDACreate2d(comm, bdX, bdY, type, M, N, m, n, dof, s, lm[], ln[], DMDA *da)

bd: Specifies boundary behavior

- DM_BOUNDARY_NONE, DM_BOUNDARY_GHOSTED, or DM_BOUNDARY_PERIODIC

type: Specifies stencil

- DMDA_STENCIL_BOX or DMDA_STENCIL_STAR

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

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

dof: Degrees of freedom per node

s: The stencil width

lm/n: Alternative array of local sizes

- Use NULL for the default

Viewing the DA

We use SNES ex5

- `ex5 -dm_view`
  - Shows both the DA and coordinate DA:
- `ex5 -dm_view draw -draw_pause -1`
- `ex5 -da_grid_x 10 -da_grid_y 10 -dm_view draw -draw_pause -1`
- `${PETSC_ARCH}/bin/mpiexec -n 4 ex5 -da_grid_x 10 -da_grid_y 10 -dm_view draw -draw_pause -1`
  - Shows PETSc numbering

DA Operators

- Evaluate only the local portion
  - No nice local array form without copies
- Use MatSetValuesStencil() to convert (i,j,k) to indices

Also use SNES ex48

- `mpiexec -n 2 ./ex5 -da_grid_x 10 -da_grid_y 10 -mat_view draw -draw_pause -1`
- `mpiexec -n 3 ./ex48 -mat_view draw -draw_pause 1 -da_refine 3 -mat_type aij`

Conclusions

PETSc can help you:

- easily construct a code to test your ideas
    - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
    - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
    - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
    - Using profiling tools and specialized implementations

Conclusions

PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations

Conclusions

PETSc can help you:

- easily construct a code to test your ideas
    - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
    - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
    - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
    - Using profiling tools and specialized implementations

Questions for Windows Users

- Have you installed cygwin?
    - Need python, make, and build-utils packages
- Will you use the GNU compilers?
    - If not, remove `link.exe`
    - If MS, check compilers from `cmd` window and use `win32fe`
- Which MPI will you use?
    - You can use `--with-mpi=0`
    - If MS, need to install MPICH2
    - If GNU, can use `--download-mpich`
- Minimal build works on Linux subsystem