

Getting Started with PETSc

Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

UNIT OBJECTIVES

- Introduce the
 Portable Extensible Toolkit for Scientific Computation
- Retrieve, Configure, Build, and Run a PETSc Example
- Empower students to learn more about PETSc

WHAT I NEED FROM YOU

- Tell me if you do not understand
- Tell me if an example does not work
- Suggest better wording or **figures**
- Followup problems at petsc-maint@mcs.anl.gov

THE ROLE OF PETSC

silver bullet.

-

THE ROLE OF PETSC

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months

-

silver bullet.

THE ROLE OF PETSC

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years)

-

silver bullet.

THE ROLE OF PETSC

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

-

silver bullet.

THE ROLE OF PETSc

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, -
silver bullet.

THE ROLE OF PETSC

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, silver bullet.

THE ROLE OF PETSC

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

WHAT IS PETSC?

- A freely available and supported research code
 - Download from <http://www.mcs.anl.gov/petsc>
 - Free for everyone, including industrial users
 - Hyperlinked manual, examples, and manual pages for all routines
 - Hundreds of tutorial-style examples
 - Support via email: petsc-maint@mcs.anl.gov
 - Usable from C, C++, Fortran 77/90, and soon Python

WHAT IS PETSc?

- Portable to any parallel system supporting MPI, including:
 - Tightly coupled systems
Cray T3E, SGI Origin, IBM SP, HP 9000, Sub Enterprise
 - Loosely coupled systems, such as networks of workstations
Compaq,HP, IBM, SGI, Sun, PCs running Linux or Windows
- PETSc History
 - Begun September 1991
 - Over 8,500 downloads since 1995 (version 2), currently 250 per month
- PETSc Funding and Support
 - Department of Energy
SciDAC, MICS Program
 - National Science Foundation
CIG, CISE, Multidisciplinary Challenge Program

NOT A “TRADITIONAL” LIBRARY

- Traditional libraries (e.g. Linpack) are used **by** an application source code: the application code creates data structures (for example arrays) fills them up (for example assembling matrices) and then pass them to the library.
- Modern libraries (e.g. PETSc) are used **with** an application code: the application source code works **with** the library creating the data structures, filling the data structures and solving.

WHO USES PETSc?

- Computational Scientists
 - PyLith (TECTON), Underworld, Columbia group
- Algorithm Developers
 - Iterative methods researchers
- Package Developers
 - SLEPc, TAO, MagPar, StGermain

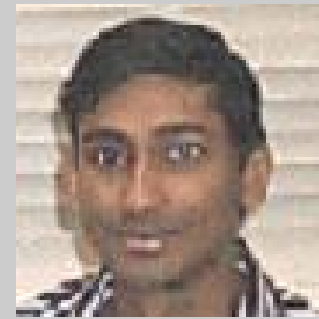
THE PETSC TEAM



Bill Gropp



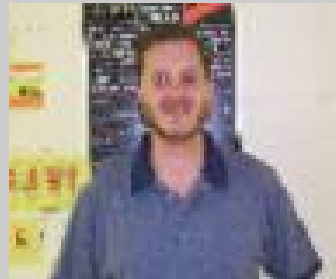
Barry Smith



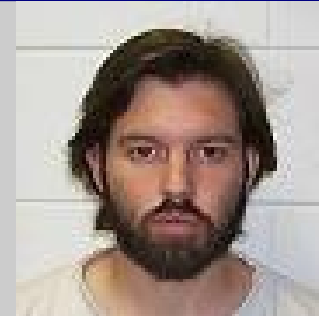
Satish Balay



Dinesh Kaushik



Kris Buschelman



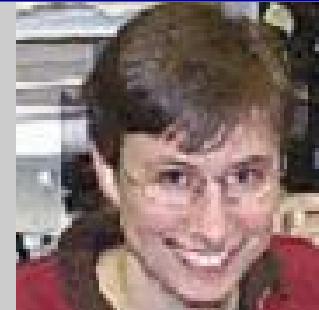
Matt Knepley



Hong Zhang



Victor Eijkhout



Lois McInnes

DOWNLOADING PETSc

- The latest tarball is on the [PETSc site](#)
 - <ftp://ftp.mcs.anl.gov/pub/petsc/petsc.tar.gz>
 - We no longer distribute patches (everything is in the distribution)
- There is a [Debian package](#)
- There is a [FreeBSD Port](#)

CLONING PETSc

- The full development repository is open to the public
 - <bk://petsc.bkbits.net/petsc-dev>
- 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

UNPACKING PETSc

- Just clone development repository

- `bk clone bk://petsc.bkbits.net/petsc-dev petsc-dev`

- `bk clone -r@Release-2.2.1 petsc-dev petsc-2.2.1`

or

- Unpack the tarball

- `tar xzf petsc.tar.gz`

EXERCISE 1

Download and Unpack PETSc!

CONFIGURING PETSC

- Set `$PETSC_DIR` to the installation root directory
- Run the configuration utility
 - `$PETSC_DIR/config/figure.py`
 - `$PETSC_DIR/config/figure.py --help`
 - `$PETSC_DIR/config/figure.py --download-mpich`
- There are many examples on the [installation page](#)
- Configuration files are placed in `$PETSC_DIR/bmake/$PETSC_ARCH`
 - `$PETSC_ARCH` has a default if not specified

CONFIGURING PETSC

- You can easily reconfigure with the same options
 - `./bmake/$PETSC_ARCH/configure.py`
- Can maintain several different configurations
 - `./config/configure.py -PETSC_ARCH=linux-fast --with-debugging=0`
- All configuration information is in `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 in PETSc
- Currently works for
 - PETSc documentation utilities (Sowing, lgrind, c2html)
 - BLAS, LAPACK, BLACS, ScaLAPACK
 - MPICH, LAM
 - ParMetis, Chaco, Jostle, Party, Scotch
 - MUMPS, Spooles, SuperLU, UMFPack
 - Prometheus, HYPRE, ML, SPAI

EXERCISE 2

Configure the PETSc that you downloaded and unpacked.

BUILDING PETSC

- Uses recursive make starting in `cd $PETSC_DIR`
 - `make`
 - Check build when done with `make test`
- Complete log for each build in `make_log_$PETSC_ARCH`
 - ALWAYS send this with bug reports
- Can build multiple configurations
 - `PETSC_ARCH=linux-fast make`
 - Libraries are in `$PETSC_DIR/lib/$PETSC_ARCH/`
- Can also build a subtree
 - `cd src/snes; make`
 - `cd src/snes; make ACTION=libfast tree`

EXERCISE 3

Build the PETSc that you configured.

EXERCISE 4

Reconfigure PETSc to use ParMetis.

1. `./bmake/linux-gnu/configure.py`
 `-PETSC_ARCH=linux-parmetis`
 `-download-parmetis`
2. `PETSC_ARCH=linux-parmetis make`
3. `PETSC_ARCH=linux-parmetis make test`

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`

USING MPI

- The **M**essage **P**assing **I**nterface is:
 - a library for parallel communication
 - a system for launching parallel jobs (mpirun/mpiexec)
 - a community standard
- Launching jobs is easy
 - `mpiexec -np 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
- Point-to-point communication
 - Happens between two processes (like in `MatMult()`)
- Reduction or scan operations
 - Happens among all processes (like in `VecDot()`)

ALTERNATIVE MEMORY MODELS

- Single process (address space) model
 - OpenMP and threads in general
 - Fortran 90/95 and compiler-discovered parallelism
 - System manages memory and (usually) thread scheduling
 - Named variables refer to the same storage
- Single name space model
 - HPF
 - Global Arrays
 - Named variables refer to the coherent values (distribution is automatic)
- Distributed memory (shared nothing)
 - Message passing
 - Names variables in different processes are unrelated

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, -vec_view_matlab, -vec_view_socket`

- Sometimes provides extra information

`-mat_view_info, -mat_view_info_detailed`

COMMON MONITORING OPTIONS

- Display the residual

`-ksp_monitor`

- Can disable dynamically

`-ksp_cancelmonitors`

- Does not display subsolvers

`-snes_monitor`

- Can display graphically

`-ksp_xmonitor`

- Can use the true residual

`-ksp_truemonitor`

- Can display different subobjects

`-snes_vecmonitor, -snes_vecmonitor_update, -snes_vecmonitor_residual`

`-ksp_gmres_krylov_monitor`

- Can display the spectrum

`-ksp_singmonitor`

EXERCISE 5

Run SNES Example 5 using some 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`

EXERCISE 6

Create a new code based upon SNES Example 5.

1. Create a new directory

```
- mkdir -p /home/knepley/proj/newsim/src
```

2. Copy the source

```
- cp ex5.c /home/knepley/proj/newsim/src
```

3. Create a PETSc makefile

```
- Add a link target
```

```
 ${CLINKER} -o $@ $< ${PETSC_SNES_LIB}
```

```
 ${FLINKER} -o $@ $< ${PETSC_FORTRAN_LIB} ${PETSC_SNES_LIB}
```

```
- include ${PETSC_DIR}/bmake/common/base
```

To get the project ready-made

```
C: bk clone bk://petsc.bkbites.net/tutorialExercise6-C newsim
```

```
F77: bk clone bk://petsc.bkbites.net/tutorialExercise6-F newsim
```

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
- High profile users
 - David Keyes
 - Rich Martineau
 - Richard Katz

Common PETSc Usage

Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

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

- Putting a breakpoint in `PetscError()` can catch errors as they are flagged
- PETSc tracks memory overwrites at the beginning and end 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
 - The `-trmalloc` option will print unfreed memory on `PetscFinalize()`

EXERCISE 1

Use the debugger to find a SEGV

Locate a memory overwrite using CHKMEMQ.

- Get the example

```
bk clone -r1.4 bk://petsc.bkbits.net/tutorialExercise1
```

- Build the example `make`
- Run it `make run` and watch the fireworks
- Run it under the debugger `make debug` and correct the error
- Build it and run again `make ex1 run` to catch the memory overwrite
- Correct the error, build it and run again `make ex1 run`

PERFORMANCE DEBUGGING

- PETSc has integrated profiling
 - The option `-log_summary` causes a report to be printed 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 and will 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

```
int stageNum;
```

```
ierr = PetscLogStageRegister(&stageNum, "name");CHKERRQ(ierr);
```

```
ierr = PetscLogStagePush(stageNum);CHKERRQ(ierr);
```

```
    code to monitor
```

```
ierr = PetscLogStagePop();CHKERRQ(ierr);
```

ADDING A LOGGING EVENT

```
static int USER_EVENT;
```

```
ierr = PetscLogEventRegister(&USER_EVENT, "name", CLASS_COOKIE);CHKERRQ(ierr)
```

```
ierr = PetscLogEventBegin(USER_EVENT,0,0,0,0);CHKERRQ(ierr);
```

```
    code to monitor
```

```
ierr = PetscLogFlops(user_event_flops);CHKERRQ(ierr);
```

```
ierr = PetscLogEventEnd(USER_EVENT,0,0,0,0);CHKERRQ(ierr);
```

ADDING A LOGGING CLASS

- Cookie identifies a class uniquely
- Initialization must happen before any objects of this type are created

```
static int CLASS_COOKIE;
```

```
ierr = PetscLogClassRegister(&CLASS_COOKIE,"name");CHKERRQ(ierr);
```

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

EFFICIENT MATRIX CREATION

- Create matrix with `MatCreate()`
- Set type with `MatSetType()`
- Determine the number of nonzeros in each row
 - loop over the grid for finite differences
 - loop over the elements for finite elements
 - need only local+ghost information
- Preallocate matrix
 - `MatSeqAIJSetPreallocation()`
 - `MatMPIAIJSetPreallocation()`

INDICATING EXPECTED NONZEROS

SEQUENTIAL SPARSE MATRICES

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

`nz`: expected number of nonzeros in any row

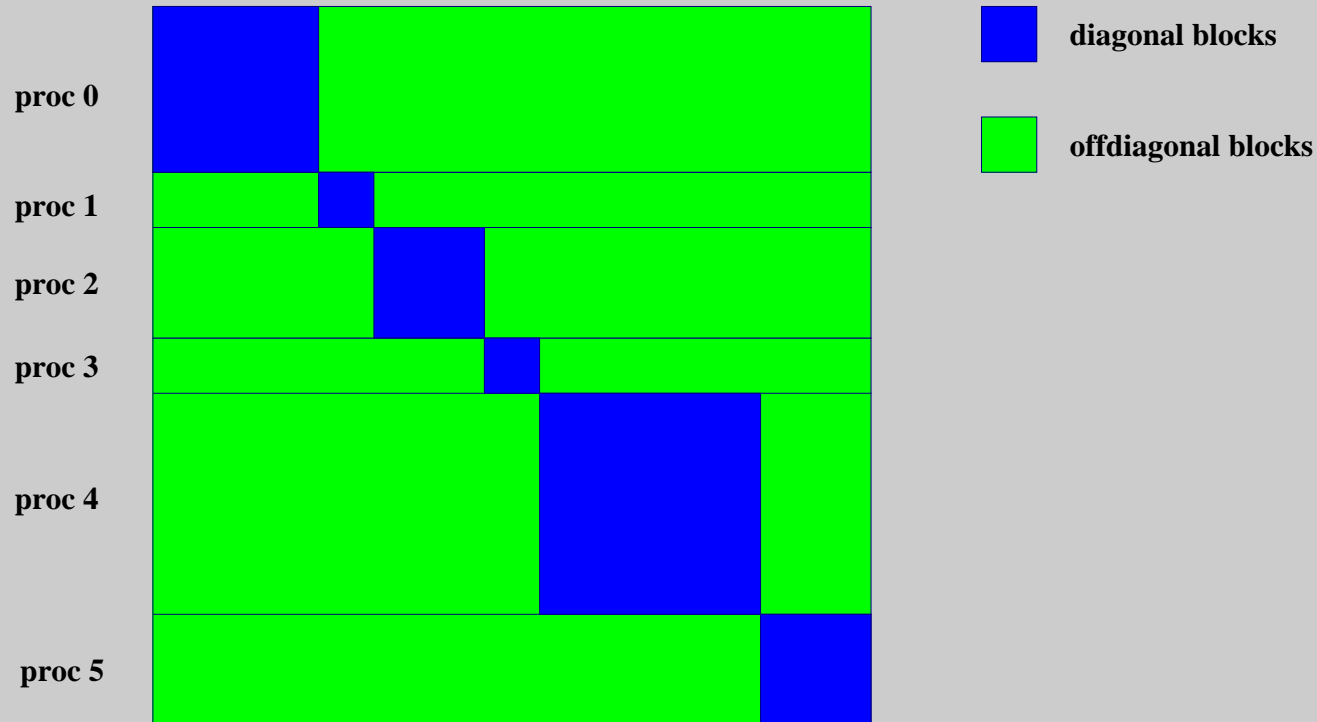
`nnz(i)`: expected number of nonzeros in row `i`



Sample Nonzero Pattern

PARALLELSPARSEMATRIX

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



- `MatGetOwnershipRange(Mat A, int *start, int *end)`

start: first locally owned row of global matrix

end-1: last locally owned row of global matrix

INDICATING EXPECTED NONZEROS

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

dnnz(i): expected number of nonzeros in row i in the diagonal block

onz: expected number of nonzeros in any row in the offdiagonal portion

onnz(i): expected number of nonzeros in row i in the offdiagonal portion

VERIFYING PREALLOCATION

- 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!
```

EXERCISE 2

Return to Unit 1:Exercise 6 and add more profiling.

- Run it `make profile` and look at the profiling report
- Add a new stage for setup
- Add a new event for `FormInitialGuess()` and log the flops
- Run it again `make ex5 profile` and look at the profiling report

PETSc Essentials

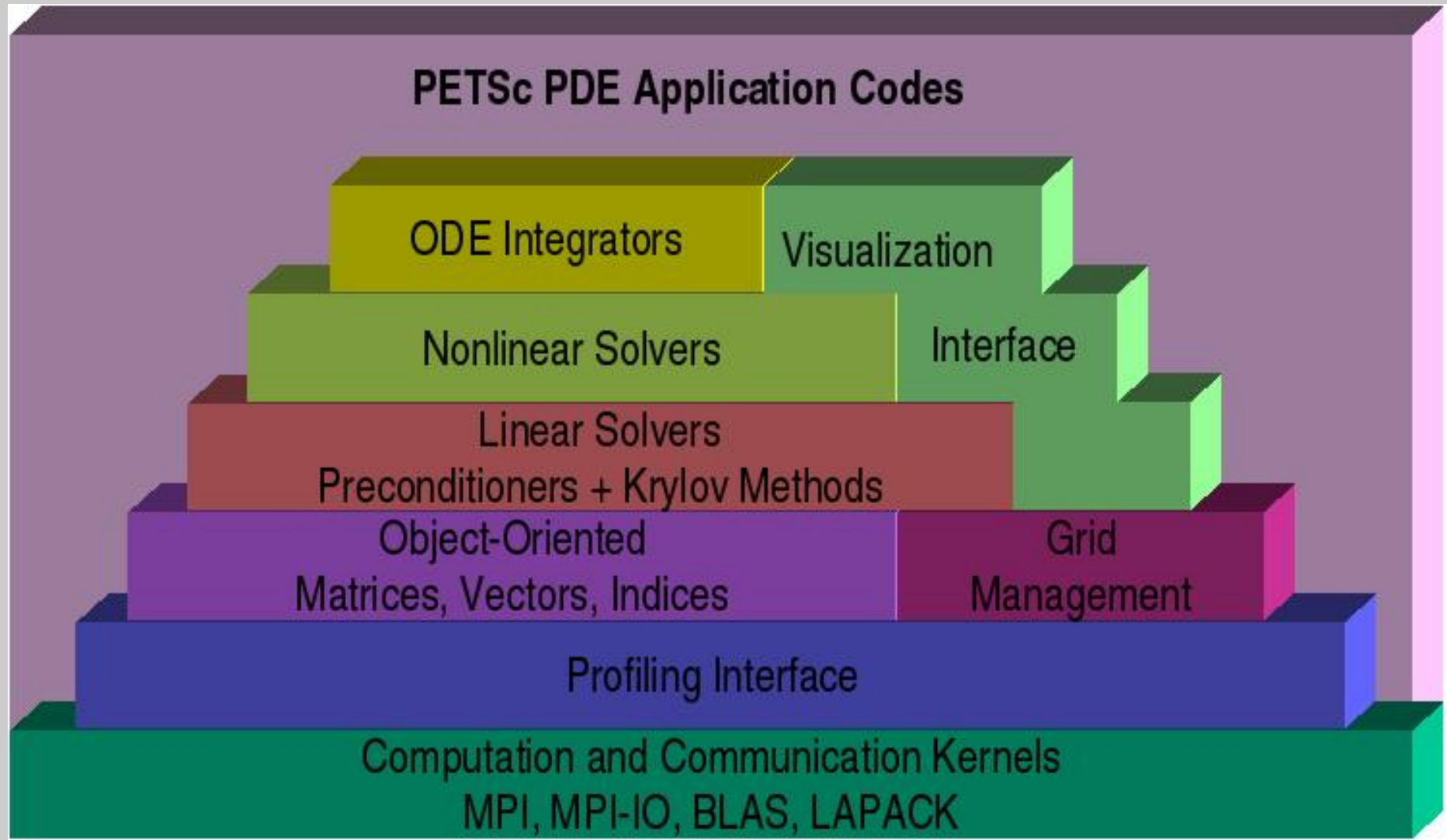
Matthew Knepley

PETSc Developer

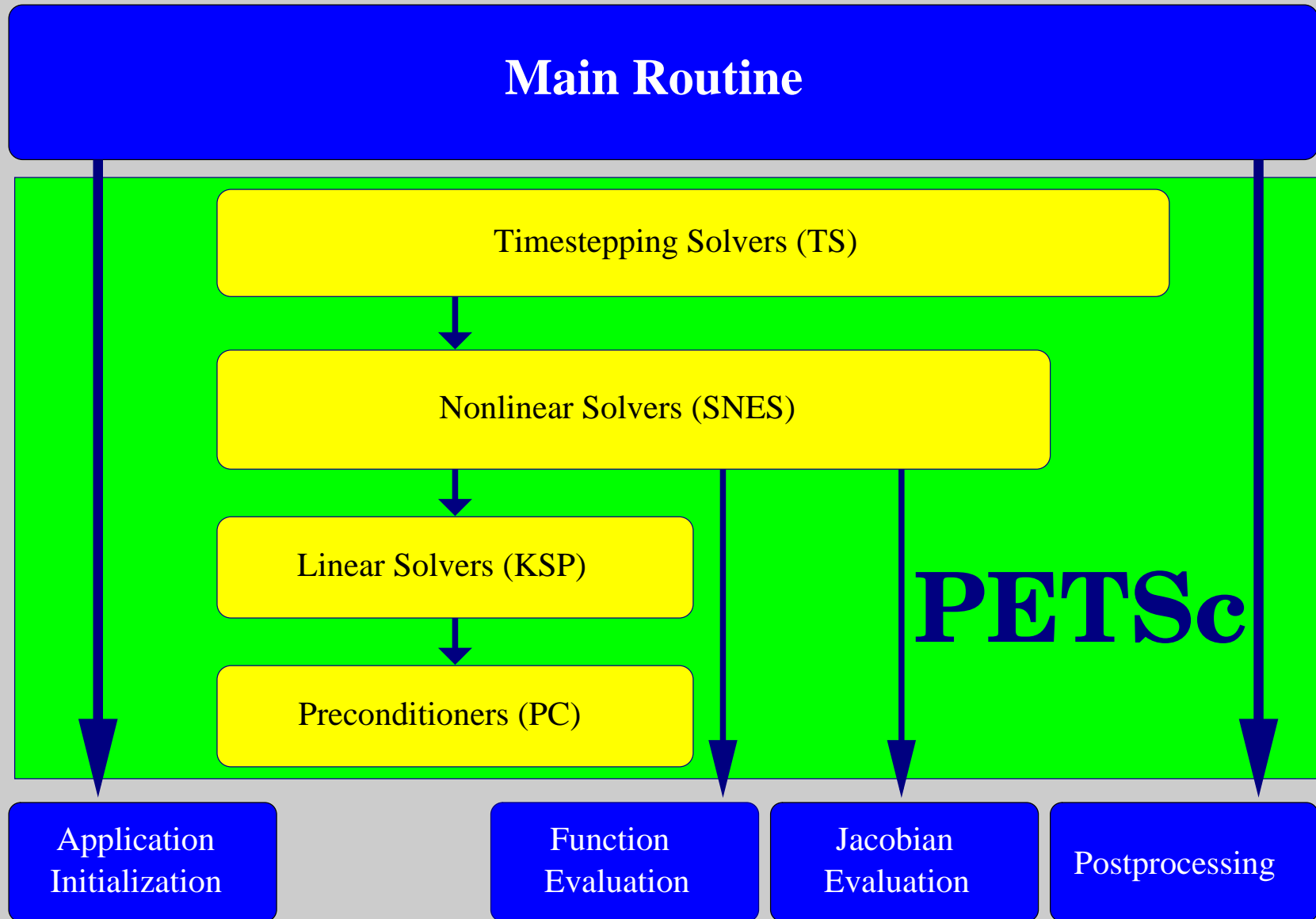
Mathematics and Computer Science Division

Argonne National Laboratory

PETSc STRUCTURE



FLOW CONTROL FOR A PETSC APPLICATION



LEVELS OF ABSTRACTION

IN MATHEMATICAL SOFTWARE

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

OBJECT-ORIENTED DESIGN

- **Design** based on operations you perform, not on the data in the object
- Example: A **vector** is
 - **not** a 1d array of numbers
 - an abstract object with addition and scalar multiplication defined
- The efficient use of the computer is an added difficulty

SYMMETRY PRINCIPLE

Interfaces to mutable data must be symmetric.

- Creation and query interfaces are paired

- Fairness

“If you can do it, your users will want to do it”

- Openness

“If you can do it, your users will want to *undo* it”

EMPIRICISM PRINCIPLE

Interfaces must allow easy testing and comparison.

- Swapping different implementations
 - “You will not be smart enough to pick the solver”
- Commonly violated in FE code
 - Elements are hard coded
- Also avoid assuming structure outside of the interface
 - Making continuous fields have certain discrete structure
 - Temptation to put metadata in a different places

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.

THE PETSC PROGRAMMING MODEL

- Goals
 - Portable, runs everywhere
 - High performance
 - Scalable parallelism
- Approach
 - Distributed memory (“shared-nothing”)
 - No special 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

COLLECTIVITY

- MPI communicators (`MPI_Comm`) specify collectivity
 - Processes involved in a computation
- Constructors are collective over a communicator
 - `VecCreate(MPI_Comm comm, Vec *x)`
 - Use `PETSC_COMM_WORLD` for all processes and `PETSC_COMM_SELF` for one
- Some operations are collective, while others are not
 - collective: `VecNorm()`
 - not collective: `VecGetLocalSize()`
- Sequences of collective calls must be in the same order on each process

WHAT IS NOT IN PETSc?

- Higher level representations of PDEs
 - Unstructured mesh generation and manipulation
 - Discretizations
 - **PETSc-CS** and **Sundance**
- Load balancing
- Sophisticated visualization capabilities
 - **MayaVi**
- Eigenvalues
 - **SLEPc**
- Optimization and sensitivity
 - **TAO** and **Veltisto**

BASIC PetscObject USAGE

Every object in PETSc supports a basic interface

Function	Operation
<code>Create()</code>	create the object
<code>Get/SetName()</code>	name the object
<code>Get/SetType()</code>	set the implementation type
<code>Get/SetOptionsPrefix()</code>	set the prefix for all options
<code>SetFromOptions()</code>	customize the object from command line options
<code>SetUp()</code>	perform other initialization
<code>View()</code>	view the object
<code>Destroy()</code>	cleanup object allocation

Also, all objects support the `-help` option.

PETSc Integration

Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

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 upgrading or seeking help
 - 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
- 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_summary` for a performance profile
 - Event timing
 - Event flops
 - Memory usage
 - MPI messages
- 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()`
- Clear, alias, reject, etc.

VECTOR ALGEBRA

What are PETSc vectors?

- Fundamental objects for storing field solutions, right-hand sides, etc.
- Each process locally owns a subvector of contiguous global data

How do I create vectors?

- `VecCreate(MPI_Comm, Vec *)`
- `VecSetSizes(Vec, int n, int N)`
- `VecSetType(Vec, VecType typeName)`
- `VecSetFromOptions(Vec)`
 - Can set the type at runtime

VECTOR ALGEBRA

A PETSc Vec

- Has a direct interface to the values
- Supports all vector space operations
 - `VecDot()`, `VecNorm()`, `VecScale()`
- Has unusual operations, e.g. `VecSqrt()`, `VecWhichBetween()`
- Communicates automatically during assembly
- Has customizable communication (scatters)

CREATING A VECTOR

```
Vec x;
```

```
PetscInt N;
```

```
PetscErrorCode ierr;
```

```
ierr = PetscInitialize(&argc, &argv, PETSC_NULL, PETSC_NULL);CHKERRQ(ierr);
```

```
ierr = PetscOptionsGetInt(PETSC_NULL, "-N", &N, PETSC_NULL);CHKERRQ(ierr);
```

```
ierr = VecCreate(PETSC_COMM_WORLD, &x);CHKERRQ(ierr);
```

```
ierr = VecSetSizes(x, PETSC_DECIDE, N);CHKERRQ(ierr);
```

```
ierr = VecSetType(x, "mpi");CHKERRQ(ierr);
```

```
ierr = VecSetFromOptions(x);CHKERRQ(ierr);
```

```
ierr = PetscFinalize();CHKERRQ(ierr);
```

PARALLEL ASSEMBLY

VECTORS AND MATRICES

- Processes may set an arbitrary entry
- 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, int n, int rows[], PetscScalar values[], mode)`
 - `mode` is either `INSERT_VALUES` or `ADD_VALUES`
- Two phase assembly allows overlap of communication and computation
 - `VecAssemblyBegin(Vec v)`
 - `VecAssemblyEnd(Vec 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) {
    for(i = 0, val = 0.0; i < N; i++, val += 10.0) {
        ierr = VecSetValues(x, 1, &i, &val, INSERT_VALUES);CHKERRQ(ierr);
    }
}
/* These two routines ensure that the data is distributed to the other processes */
ierr = VecAssemblyBegin(x);CHKERRQ(ierr);
ierr = VecAssemblyEnd(x);CHKERRQ(ierr);
```

A BETTER WAY TO SET THE ELEMENTS OF A VECTOR

```
ierr = VecGetOwnershipRange(x, &low, &high);CHKERRQ(ierr);  
for(i = low, val = low*10.0; i < high; i++, val += 10.0) {  
    ierr = VecSetValues(x, 1, &i, &val, INSERT_VALUES);CHKERRQ(ierr);  
}  
/* These two routines ensure that the data is distributed to the other processes */  
ierr = VecAssemblyBegin(x);CHKERRQ(ierr);  
ierr = VecAssemblyEnd(x);CHKERRQ(ierr);
```

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 PETSc `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 inexpensive and do not involve a copy

VECGETARRAY IN C

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

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

VECGETARRAY IN F77

```
#include "petsc.h"
#include "petscvec.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 "petsc.h"  
#include "petscvec.h"  
    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)
```

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 Diagonal, etc.
- Supports structures for many packages
 - MUMPS, Spooles, SuperLU, UMFPack, DSCPack

HOW DO I CREATE MATRICES?

- `MatCreate(MPI_Comm, Mat *)`
- `MatSetSizes(Mat, int m, int n, int M, int N)`
- `MatSetType(Mat, MatType typeName)`
- `MatSetFromOptions(Mat)`
 - Can set the type at runtime
- `MatSetValues(Mat, ...)`
 - **MUST** be used, but does automatic communication

MATRIX POLYMORPHISM

The PETSc Mat has a single user interface,

- Matrix assembly
 - `MatSetValues()`
- 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(Mat m, 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(Mat m, type)`
 - `MatAssemblyEnd(Mat m, type)`
 - `type` is either `MAT_FLUSH_ASSEMBLY` or `MAT_FINAL_ASSEMBLY`

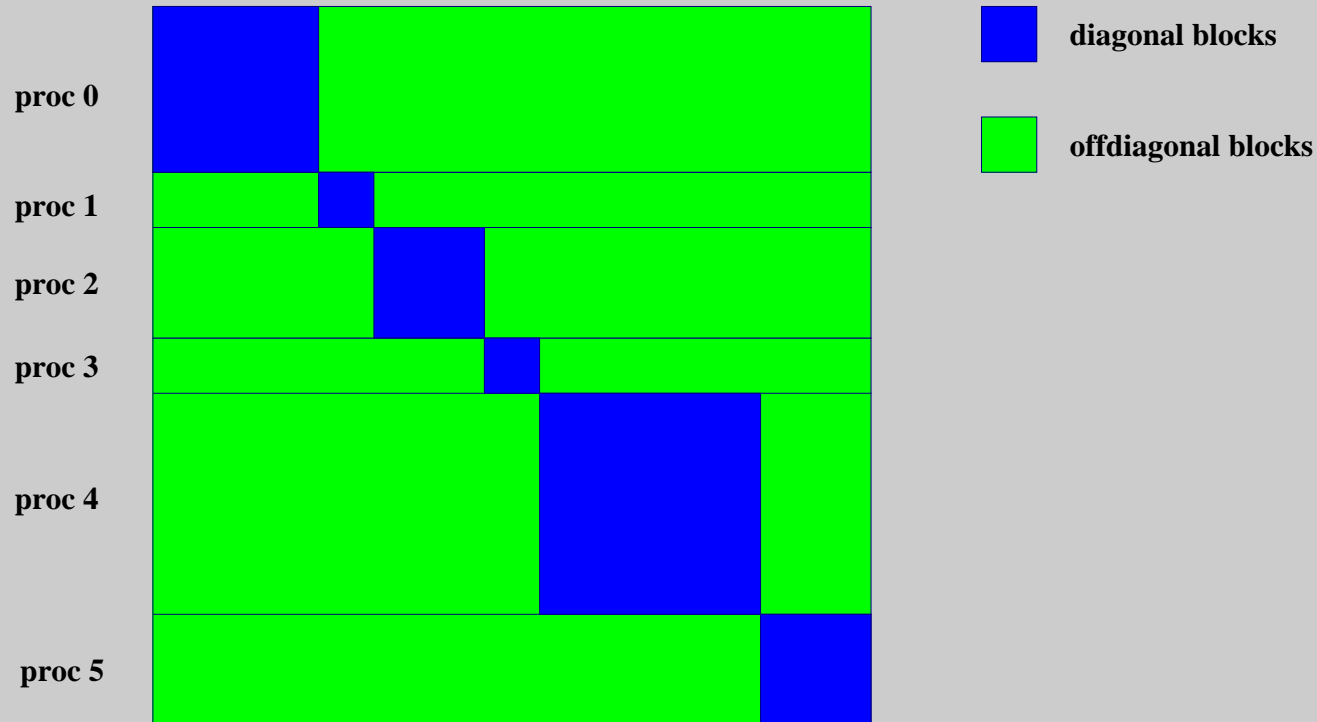
ONE WAY TO SET THE ELEMENTS OF A MATRIX

SIMPLE 3-POINT STENCIL FOR 1D LAPLACIAN

```
values[0] = -1.0; values[1] = 2.0; values[2] = -1.0;
if (rank == 0) { /* Only one process creates matrix */
    for(row = 0; row < N; row++) {
        cols[0] = row-1; cols[1] = row; cols[2] = row+1;
        if (row == 0) {
            ierr = MatSetValues(A, 1, &row, 2, &cols[1], &values[1], INSERT_VALUES);CHKERR(ierr);
        } else if (row == N-1) {
            ierr = MatSetValues(A, 1, &row, 2, cols, values, INSERT_VALUES);CHKERRQ(ierr);
        } else {
            ierr = MatSetValues(A, 1, &row, 3, cols, values, INSERT_VALUES);CHKERRQ(ierr);
        }
    }
}
ierr = MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
ierr = MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);
```

PARALLELSPARSEMATRIX

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



- `MatGetOwnershipRange(Mat A, int *start, int *end)`

start: first locally owned row of global matrix

end-1: last locally owned row of global matrix

A BETTER WAY TO SET THE ELEMENTS OF A MATRIX

SIMPLE 3-POINT STENCIL FOR 1D LAPLACIAN

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

WHY ARE PETSC MATRICES THAT WAY?

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

SOLVERS

- **Explicit:**
 - Field variables are updated using local neighbor information
- **Semi-implicit:**
 - Some subsets of variables are updated with global solves
 - Others with direct local updates
- **Implicit:**
 - Most or all variables are updated in a single global solve

LINEAR SOLVERS

KRYLOV METHODS

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

NONLINEAR SOLVERS

NEWTON METHODS

- Using PETSc linear algebra, just add:
 - `SNESSetFunction(SNES snes, Vec r, residualFunc, void *ctx)`
 - `SNESSetJacobian(SNES snes, Mat A, Mat M, jacFunc, void *ctx)`
 - `SNESsolve(SNES snes, Vec b, Vec x)`
- Can access subobjects
 - `SNESGetKSP(SNES snes, KSP *ksp)`
- Can customize subobjects from the cmd line
 - Set the subdomain preconditioner to ILU with `-sub_pc_type ilu`

BASIC SOLVER USAGE

We will illustrate basic solver usage with SNES.

- Use `SNESSetFromOptions()` so that that everything is set dynamically
 - Use `-snes_type` to set the 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

1. Sequential LU

- ILUDT (SPARSEKIT2, Yousef Saad, U of MN)
- EUCLID & PILUT (Hypre, David Hysom, LLNL)
- ESSL (IBM)
- SuperLU (Jim Demmel and Sherry Li, LBNL)
- Matlab
- UMFPACK (Tim Davis, U. of Florida)
- LUSOL (MINOS, Michael Saunders, Stanford)

2. Parallel LU

- MUMPS (Patrick Amestoy, IRIT)
- SPOOLES (Cleve Ashcroft, Boeing)
- SuperLU_Dist (Jim Demmel and Sherry Li, LBNL)

3. Parallel Cholesky

- DSCPACK (Padma Raghavan, Penn. State)

4. XYTlib - parallel direct solver (Paul Fischer and Henry Tufo, ANL)

3RD PARTY PRECONDITIONERS IN PETSC

1. Parallel ICC

- BlockSolve95 (Mark Jones and Paul Plassman, ANL)

2. Parallel ILU

- BlockSolve95 (Mark Jones and Paul Plassman, ANL)

3. Parallel Sparse Approximate Inverse

- Parasails (Hypre, Edmund Chow, LLNL)
- SPAI 3.0 (Marcus Grote and Barnard, NYU)

4. Sequential Algebraic Multigrid

- RAMG (John Ruge and Klaus Steuben, GMD)
- SAMG (Klaus Steuben, GMD)

5. Parallel Algebraic Multigrid

- Prometheus (Mark Adams, PPPL)
- BoomerAMG (Hypre, LLNL)
- ML (Trilinos, Ray Tuminaro and Jonathan Hu, SNL)

HIGHER LEVEL ABSTRACTIONS

The PETSc DA class is a topology interface.

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

The PETSc DM class is a hierarchy interface.

- Supports multigrid
 - DMMG combines it with the MG preconditioner
- Abstracts the logic of multilevel methods

Advanced PETSc

Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

SNESCALLBACKS

The SNES interface is based upon callback functions

- `SNESSetFunction()`
- `SNESSetJacobian()`

When PETSc needs to evaluate the nonlinear residual $F(x)$, the solver calls the **user's** function inside the application. The user function get application state through the `ctx` variable. PETSc never sees application data.

SNES FUNCTION

The user provided function which calculates the nonlinear residual has signature

```
PetscErrorCode (*func)(SNES snes, Vec x, Vec r, void *ctx)
```

x: The current solution

r: The residual

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

- Use this to pass application information, e.g. physical constants

SNES JACOBIAN

The user provided function which calculates the Jacobian has signature

```
PetscErrorCode (*func)(SNES snes, Vec x, Mat *J, Mat *M,  
                        MatStructure *flag, void *ctx)
```

x: The current solution

J: The Jacobian

M: The Jacobian preconditioning matrix (possibly J itself)

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

- Use this to pass application information, e.g. physical constants

- Possible `MatrStructure` values are:

- `SAME_NONZERO_PATTERN`, `DIFFERENT_NONZERO_PATTERN`, ...

Alternatively, you can use

- a builtin sparse finite difference approximation

- automatic differentiation

- AD support via `ADIC/ADIFOR` (P. Hovland and B. Norris from ANL)

SNES VARIANTS

- Line search strategies
- Trust region approaches
- Pseudo-transient continuation
- Matrix-free variants

FINITE DIFFERENCE JACOBIANS

PETSc can compute and explicitly store a Jacobian via 1st-order FD

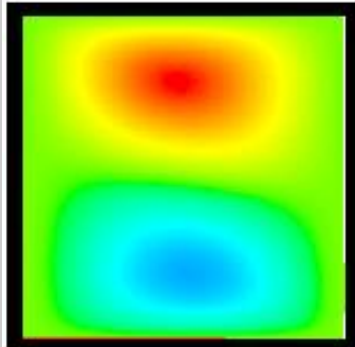
- Dense
 - Activated by `-snes_fd`
 - Computed by `SNESDefaultComputeJacobian()`
- Sparse via colorings
 - Coloring is created by `MatFDColoringCreate()`
 - Computed by `SNESDefaultComputeJacobianColor()`

Can also use Matrix-free Newton-Krylov via 1st-order FD

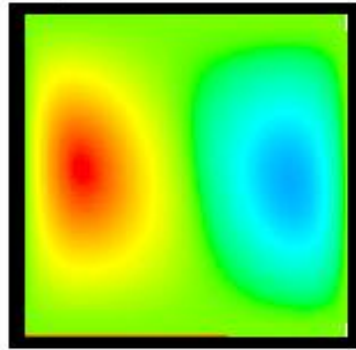
- Activated by `-snes_mf` without preconditioning
- Activated by `-snes_mf_operator` with user-defined preconditioning
 - Uses preconditioning matrix from `SNESSetJacobian()`

SNES EXAMPLE: DRIVEN CAVITY

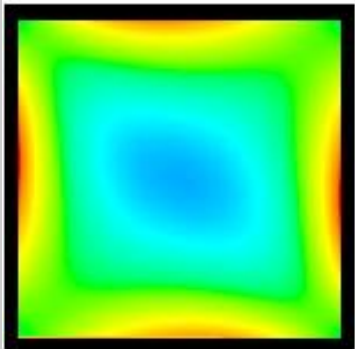
Solution Components



velocity: u



velocity: v



vorticity:



temperature: T

- Velocity-vorticity formulation
- Flow driven by lid and/or bouyancy
- Logically regular grid
 - Parallelized with DA
- Finite difference discretization
- Authored by David Keyes

`$PETCS_DIR/src/snes/examples/tutorials/ex19.c`

DRIVEN CAVITY APPLICATION CONTEXT

```
typedef struct {  
    /*--- basic application data ---*/  
    double lid_velocity;    /* Velocity of the lid */  
    double prandtl, grashof; /* Prandtl and Grashof numbers */  
    int mx, my;            /* Grid points in x and y */  
    int mc;                /* Degrees of freedom per node */  
    PetscTruth draw_contours; /* Flag for drawing contours */  
    /*--- parallel data ---*/  
    MPI_Comm comm;        /* Communicator */  
    DA da;                /* Distributed array */  
    Vec localX, localF;   /* Local ghosted solution and residual */  
} AppCtx;
```

\$PETCS_DIR/src/snes/examples/tutorials/ex19.c

DRIVEN CAVITY RESIDUAL EVALUATION

```
DrivenCavityFunction(SNES snes, Vec X, Vec F, void *ptr)
{
  AppCtx      *user = (AppCtx *) ptr;
  int         istart, iend, jstart, jend; /* local starting and ending grid points */
  PetscScalar *f;                        /* local vector data */
  PetscReal   grashof = user->grashof;
  PetscReal   prandtl = user->prandtl;
  PetscErrorCode ierr;

  /* Not Shown: Code to communicate nonlocal ghost point data (scatters) */
  ierr = VecGetArray(F, &f); CHKERRQ(ierr);
  /* Not Shown: Code to compute local function components */
  ierr = VecRestoreArray(F, &f); CHKERRQ(ierr);
  return 0;
}
```

\$PETCS_DIR/src/snes/examples/tutorials/ex19.c

BETTER DRIVEN CAVITY RESIDUAL EVALUATION

```
PetscErrorCode DrivenCavityFuncLocal(DALocalInfo *info,Field **x,Field **f,void *ctx)
{
    /* Not Shown: Handle boundaries */
    /* Compute over the interior points */
    for(j = info->ys; j < info->xs+info->xm; j++) {
        for(i = info->xs; i < info->ys+info->ym; i++) {
            /* Not Shown: convective coefficients for upwinding */
            /* U velocity */
            u          = x[j][i].u;
            uxx        = (2.0*u - x[j][i-1].u - x[j][i+1].u)*hydhx;
            uyy        = (2.0*u - x[j-1][i].u - x[j+1][i].u)*hxdhy;
            f[j][i].u  = uxx + uyy - .5*(x[j+1][i].omega-x[j-1][i].omega)*hx;
            /* Not Shown: V velocity, Omega, Temperature */
        }
    }
}
```

WHAT IS A DA?

DA is a topology interface that handles parallel data layout on structured grids

- Handles local and global indices
 - `DAGetGlobalIndices()` and `DAGetAO()`
- Provides local and global vectors
 - `DAGetGlobalVector()` and `DAGetLocalVector()`
- Handles ghost values coherence
 - `DAGetGlobalToLocal()` and `DAGetLocalToGlobal()`

CREATING A DA

```
DACreate1d(comm, DAPeriodicType wrap, M, dof, s, lm[], DA *inra)
```

`wrap`: Specifies periodicity

- `DA_NONPERIODIC` or `DA_XPERIODIC`

`M`: Number of grid points in x-direction

`dof`: Degrees of freedom per node

`s`: The stencil width

`lm`: Alternative array of local sizes

- Use `PETSC_NULL` for the default

CREATING A DA

```
DACreate2d(comm, wrap, type, M, N, m, n, dof, s, lm[], ln[], DA *inra)
```

`wrap`: Specifies periodicity

- `DA_NONPERIODIC`, `DA_XPERIODIC`, `DA_YPERIODIC`, or `DA_XYPERIODIC`

`type`: Specifies stencil

- `DA_STENCIL_BOX` or `DA_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

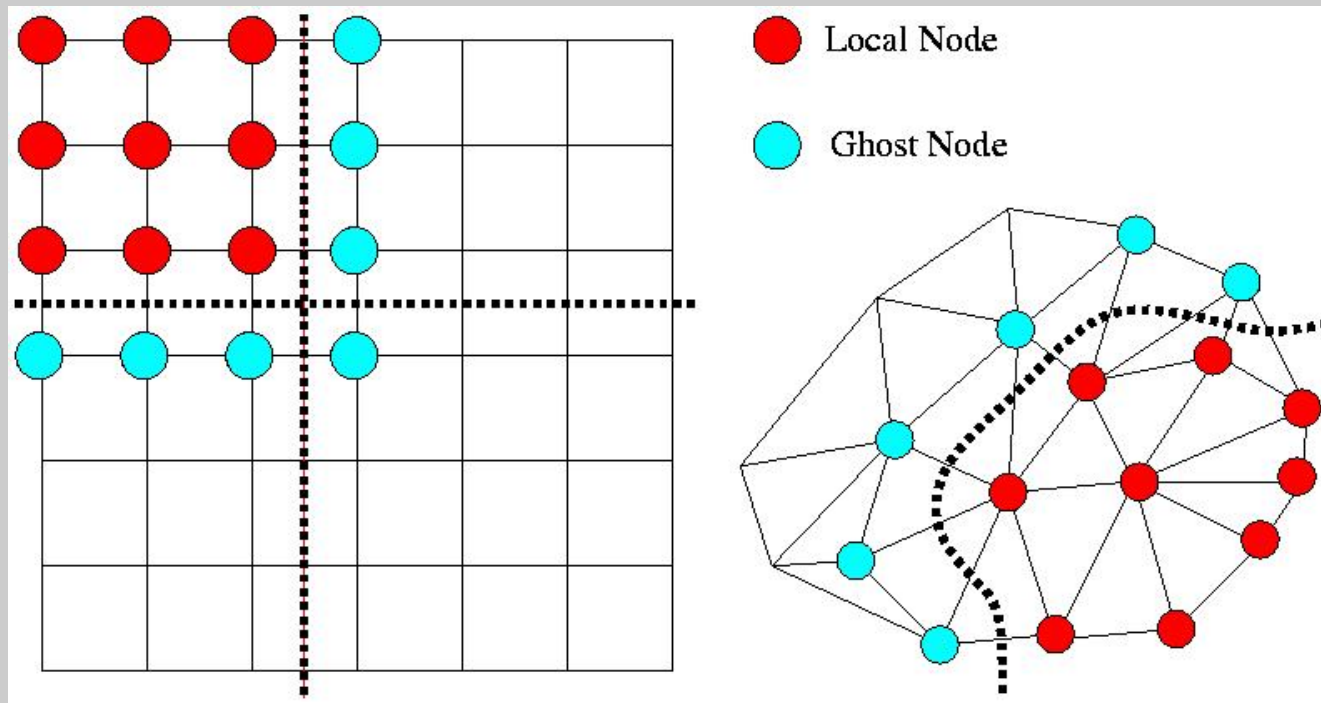
`lm/n`: Alternative array of local sizes

- Use `PETSC_NULL` for the default

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



DA 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

DA GLOBAL VS. LOCAL NUMBERING

- **Global:** Each vertex belongs to a unique process and has a unique id
- **Local:** Numbering includes **ghost** vertices from neighboring processes

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

DA VECTORS

- The DA 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
 - `DACreateGlobalVector(DA da, Vec *gvec)`
- Local vectors are sequential (and usually temporary)
 - Each process stores its local portion plus ghost values
 - `DACreateLocalVector(DA da, Vec *lvec)`
 - includes ghost values!

UPDATING GHOSTS

Two-step process enables overlapping computation and communication

- `DAGlobalToLocalBegin(da, gvec, mode, lvec)`
 - `gvec` provides the data
 - `mode` is either `INSERT_VALUES` or `ADD_VALUES`
 - `lvec` holds the local and ghost values
- `DAGlobalToLocal End(da, gvec, mode, lvec)`
 - Finishes the communication

The process can be reversed with `DALocalToGlobal()`.

DA LOCAL FUNCTION

The user provided function which calculates the nonlinear residual in 2D has signature

```
PetscErrorCode (*lfunc)(DALocalInfo *info, PetscScalar **x,  
                        PetscScalar **r, void *ctx)
```

info: All layout and numbering information

x: The current solution

- Notice that it is a multidimensional array

r: The residual

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

The local DA function is activated by calling

```
SNESSetFunction(snes, r, SNESDAFormFunction, ctx)
```

DA LOCAL JACOBIAN

The user provided function which calculates the nonlinear residual in 2D has signature

```
PetscErrorCode (*lfunc)(DALocalInfo *info, PetscScalar **x, Mat J,  
                        void *ctx)
```

info: All layout and numbering information

x: The current solution

J: The Jacobian

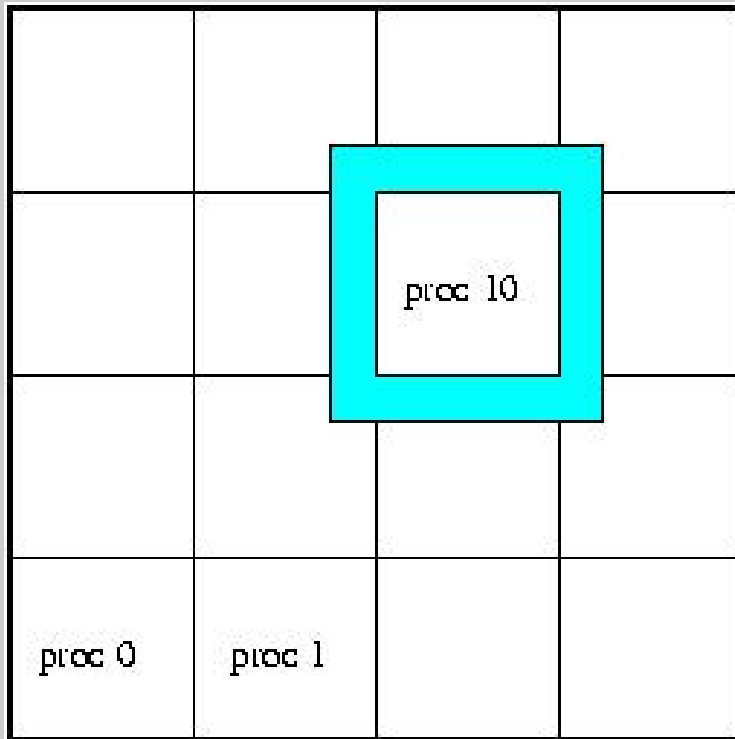
ctx: The user context passed to `DASetLocalFunction()`

The local DA function is activated by calling

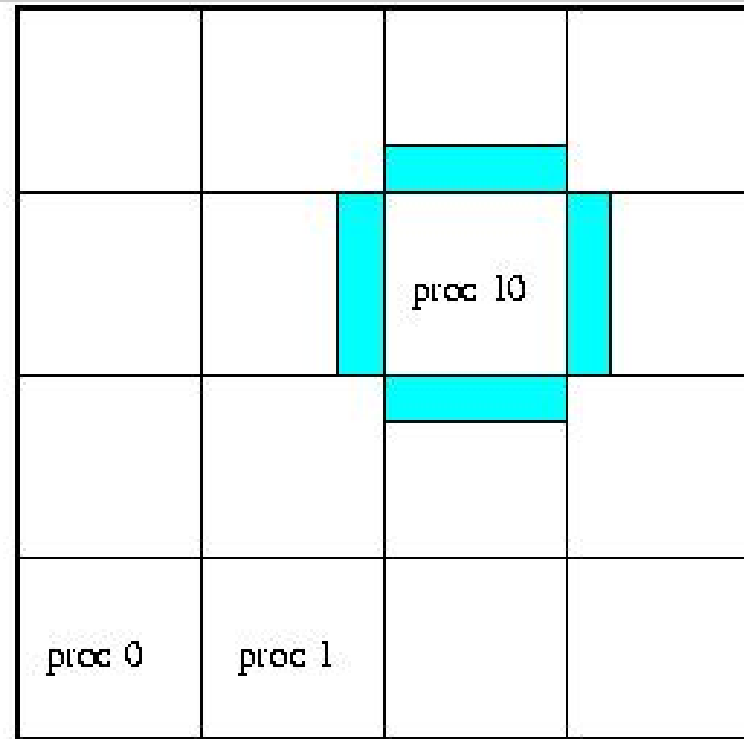
```
SNESSetJacobian(snes, J, J, SNESDASComputeJacobian, ctx)
```

DA STENCILS

Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

SETTING VALUES FOR FINITE DIFFERENCES

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n, MatStencil  
                    idxn[], values[], 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 rows and columns

MAPPING BETWEEN GLOBAL NUMBERINGS

- Natural global numbering
 - Convenient for visualization, boundary conditions, etc.
- Convert between global numbering schemes using `AO`
 - `DAGetAO(DA da, AO *ao)`
- Handled automatically by some utilities (e.g., `VecView()`) for `DA` vectors

DA EXAMPLE: BRATU

- Create SNES and DA
- Use `DASetLocalFunction()` and `DASetLocalJacobian()` to set user callbacks
 - Use `DAGetMatrix()` to get DA matrix for SNES
- Use `SNESDAFormFunction()` and `SNESDAComputeJacobian()` for SNES callback
 - Could also use `FormFunctionMatlab()`
 - Could also use `SNESDefaultComputeJacobian()`

`$PETCS_DIR/src/snes/examples/tutorials/ex5.c`

DA EXAMPLE: BRATU

```
PetscErrorCode LocalFunc(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);
            }
        }
    }
}
```

\$PETCS_DIR/src/snes/examples/tutorials/ex5.c

DA EXAMPLE: BRATU

```
int LocalJac(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;
                ierr = MatSetValuesStencil(jac, 1, &row, 1, &row, v, INSERT_VALUES);CHKERRQ(ierr);
            } 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;
                ierr = MatSetValuesStencil(jac, 1, &row, 5, col, v, INSERT_VALUES);CHKERRQ(ierr);
            }
        }
    }
}
```

A Poisson Problem

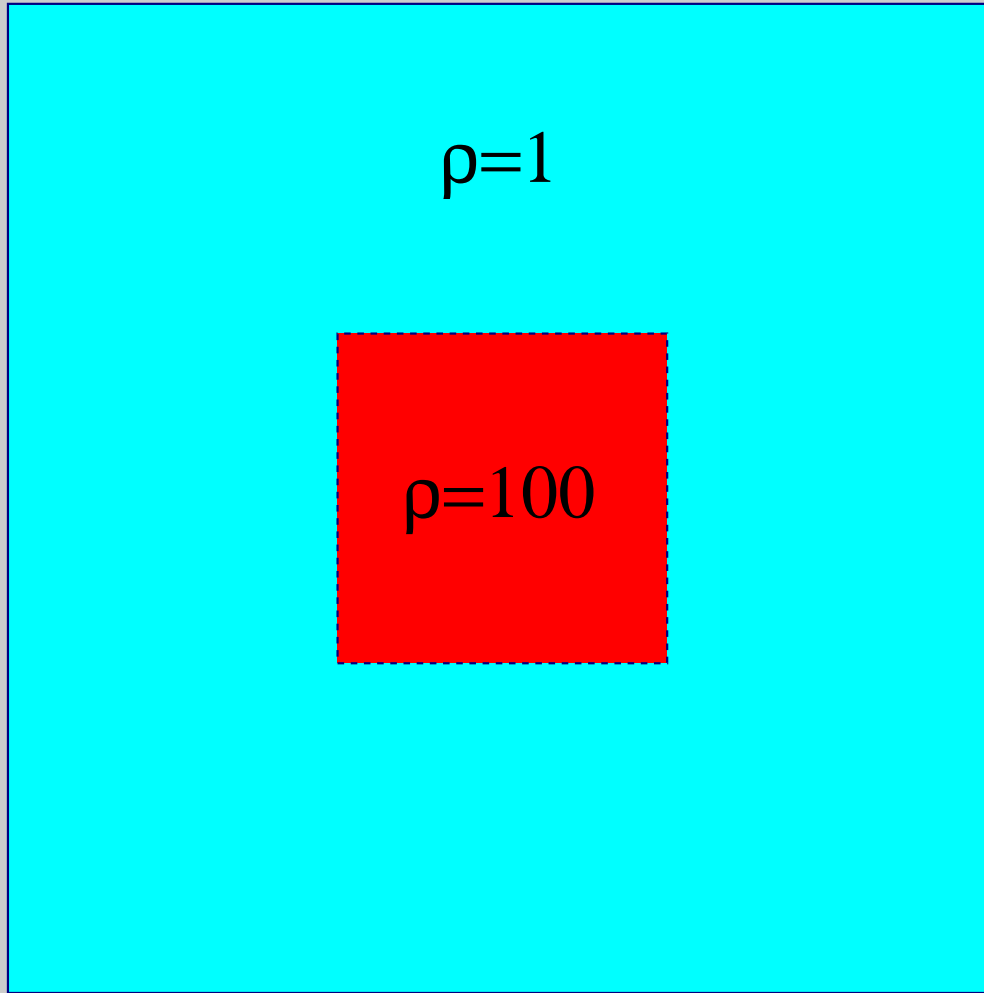
Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

PROBLEM DOMAIN



PROBLEM STATEMENT

Inhomogeneous Laplacian in 2D. Modeled by the partial differential equation

$$\nabla \cdot (\rho \nabla \mathbf{u}) = \mathbf{f} \quad \text{on} \quad \Omega = [0, 1] \times [0, 1],$$

with forcing function

$$\mathbf{f}(x, y) = e^{-(1-x)^2/\nu} e^{-(1-y)^2/\nu},$$

with Dirichlet boundary conditions

$$\mathbf{u} = \mathbf{f}(x, y) \quad \text{on} \quad \partial\Omega,$$

or homogeneous Neumann boundary conditions

$$\hat{n} \cdot \nabla \mathbf{u} = 0 \quad \text{on} \quad \partial\Omega.$$

REVISION 1.1

For the initial try, we modify a common PETSc DMMG skeleton:

- Use a simple FD 5-point stencil discretization
- Use a structured grid (DA)
- Use a hierarchical method (DMMG)
- Only implement Dirichlet BC (simple masking)

REVISION 1.2

Now utilize some more PETSc features:

- Add `UserContext` structure to hold ν and the BC type
 - Need to set the context at each DM level
- Add Neumann BC using a `MatNullSpace`
 - Used to project onto the orthogonal complement
 - `KSPSetNullSpace()`
- Set parameters from the command line
 - `PetscOptionsBegin()`, `PetscOptionsEnd()`
 - `PetscOptionsScalar()`, `PetscOptionsString()`
 - By hand, `PetscOptionsGetScalar()`, `PetscOptionsGetString()`
- Fixed scaling for anisotropic grids

REVISION 1.6

Barry fixed the example to converge nicely:

- Set nullspace on all DM levels
 - Actually set in the smoother (KSP)
 - Same idea as the user context
 - Now completely handled by `DMMGSetNullSpace()`
- Remove the null space component of the rhs
 - `MatNullSpaceRemove()`
 - Usually handled by the model
- Add a shift to the coarse grid LU for Neumann BC
 - System is singular so augment with the identity
 - One extra step if coarse solve is redundant
- Fix weighting for center point of Neumann condition
 - Depends on the number of missing points
- Also use `PetscOptionsElist()` to set BC
 - Can provide a nice listbox using the GUI

DMMG GRIDS

The user specifies the **coarse** grid, and then DMMG successively refines it.

- In our problem, we begin with a 3×3 grid
 - We LU factor a 9×9 matrix
- By level 10, we have a 1025×1025 grid
 - Our final solution has 1,050,625 unknowns
- Set the initial grid using `-da_grid_x` and `-da_grid_y`
- Set the number of levels using `-dmmg_nlevels`

MESH INDEPENDENCE

The iteration number should be independent of the mesh size, or the number of levels.

Levels	Unknowns	KSP Iterations
2	25	2
4	289	2
6	4225	3
8	66049	2
10	1050625	2

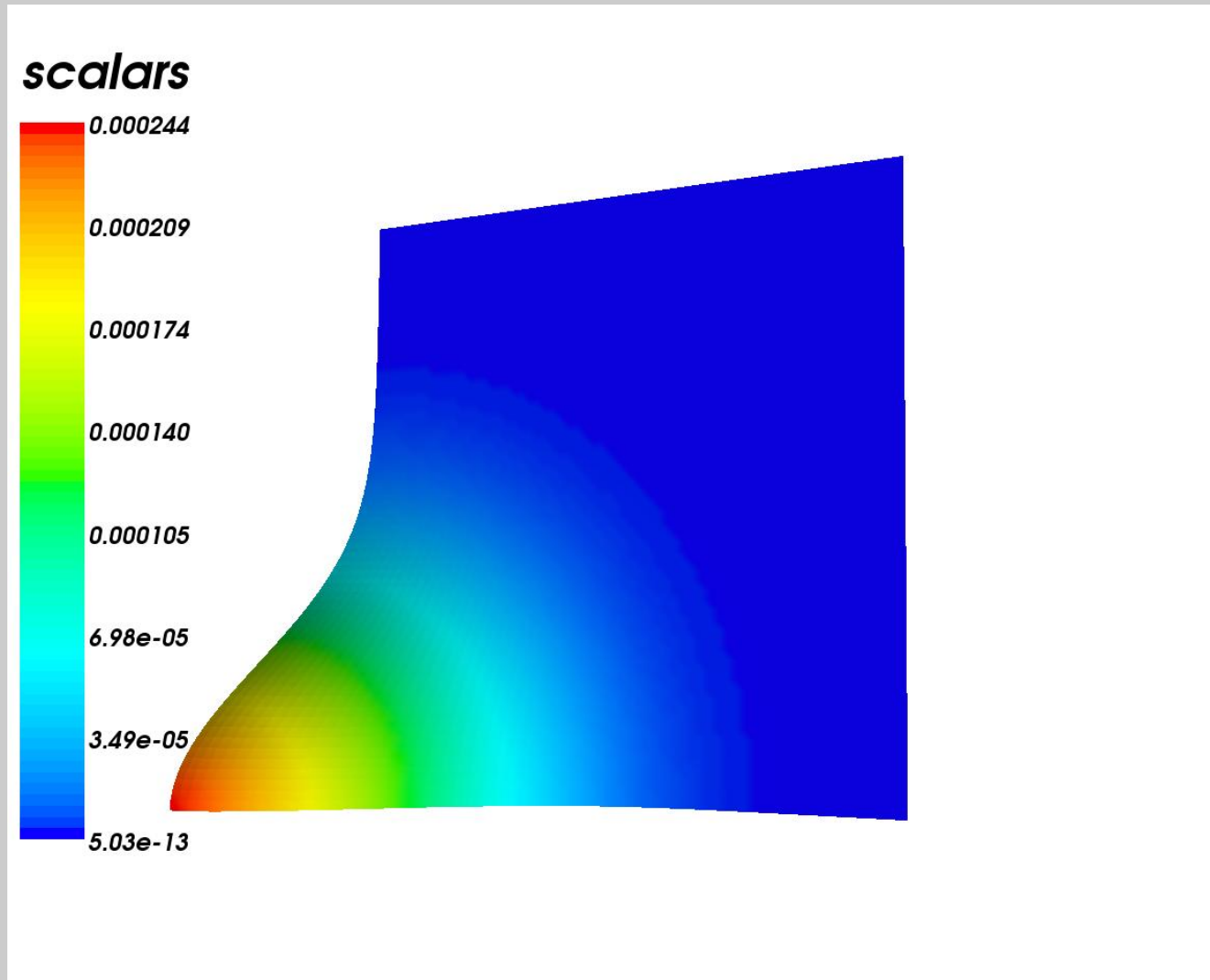
We have used Dirichlet BC above

VIEWING THE DA

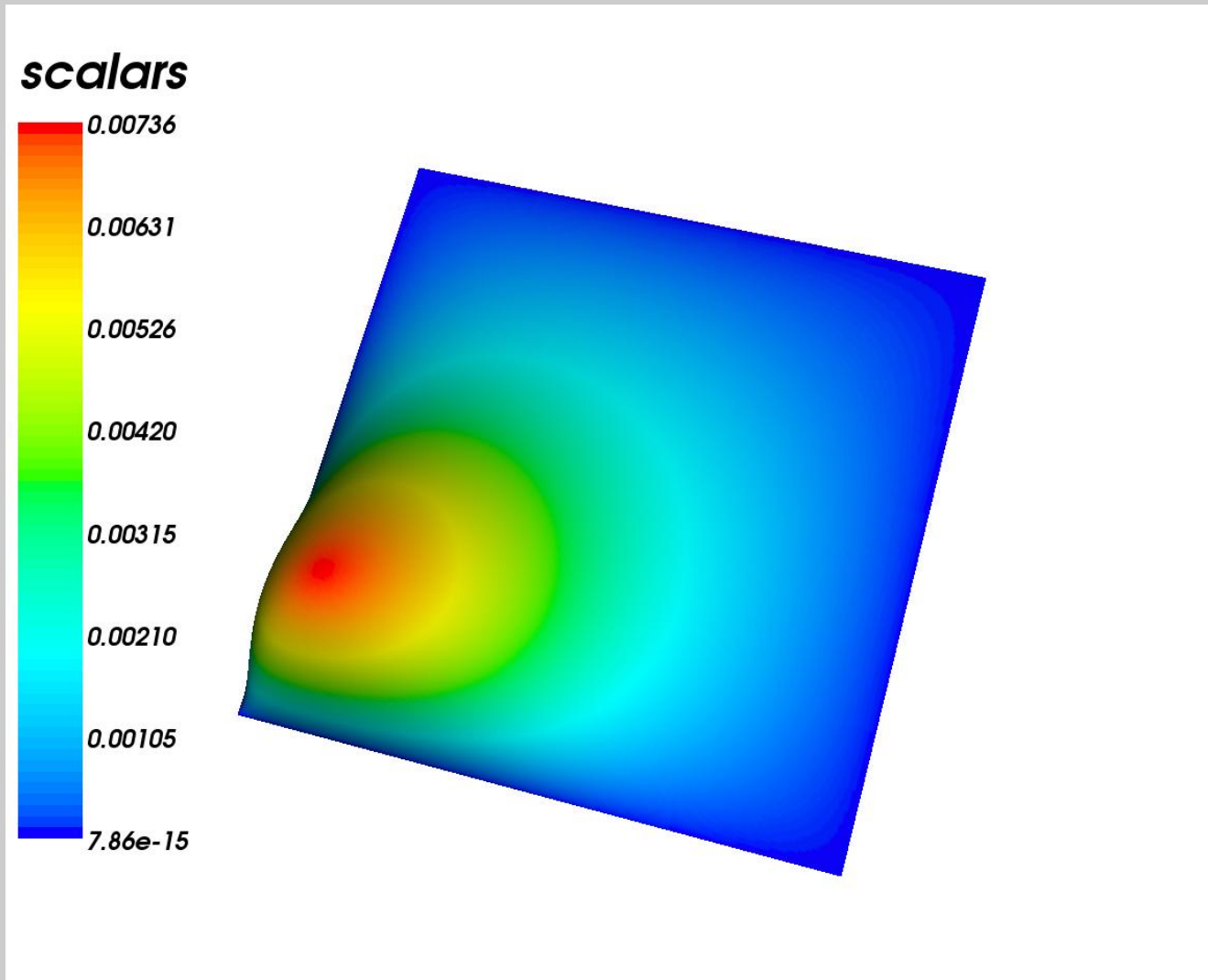
- `-da_view` outputs an ascii description
- `-da_view_draw` display the grid
 - First the grid itself is drawn
 - Global variable numbers are then provided
 - Finally ghost variable numbers are shown for error checking
- `-vec_view_draw` draws a contour plot for DA vectors
 - The contour grid can be shown with `-draw_contour_grid`

FORCING FUNCTION

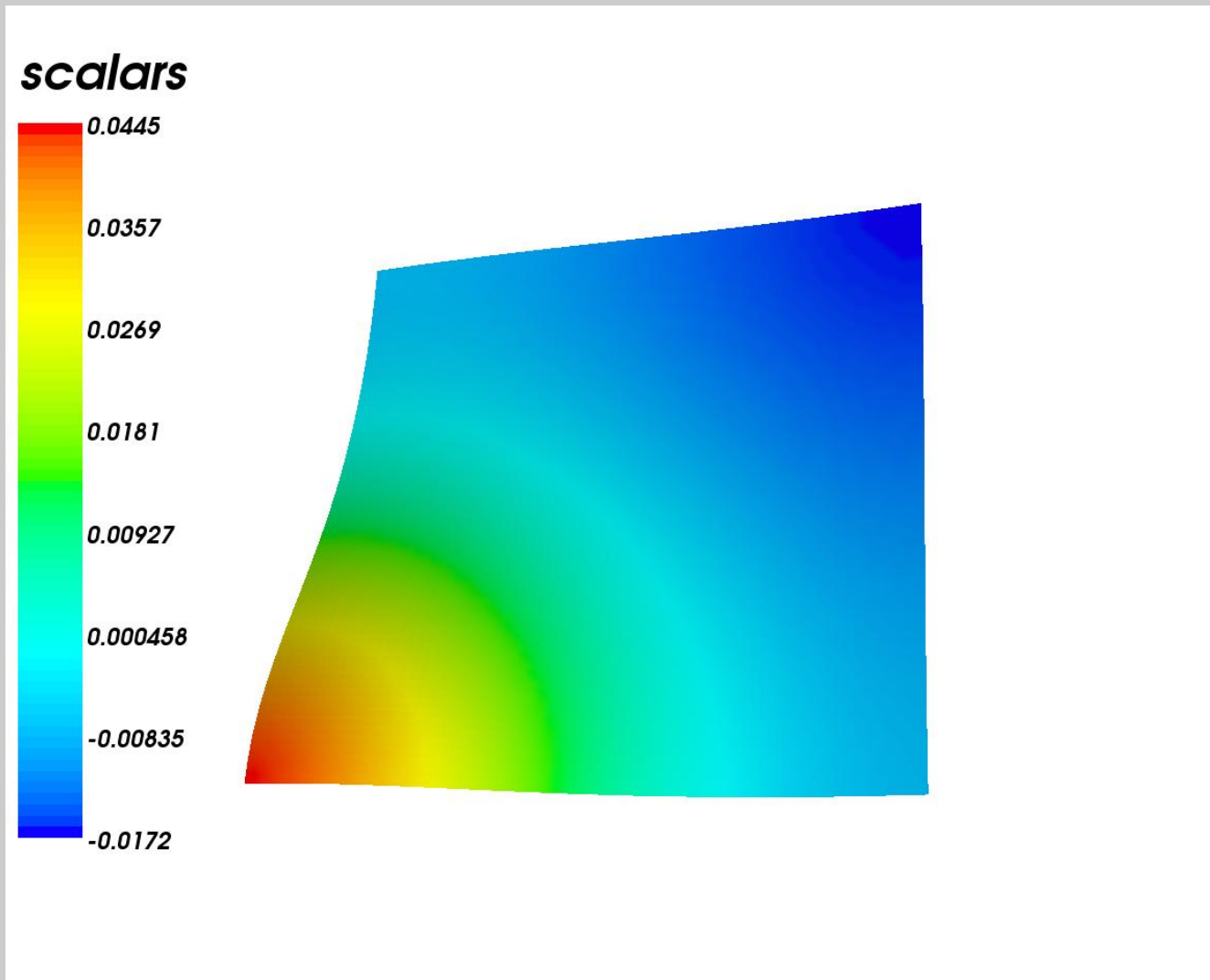
The rhs of our linear system drives the solution:



DIRICHLET SOLUTION



NEUMANN SOLUTION



MULTIGRID OPTIONS

- Choose V-cycle or W-cycle using `-pc_mg_cycles`
- Can set the iteration method using `-pc_mg_type`
 - MULTIPLICATIVE, ADDITIVE, FULL, KASKADE
- Choose the number of steps in both the up and down smoothers
 - `-pc_mg_smoothup`, `-pc_mg_smoothdown`
- The coarse problem has prefix `mg_coarse_`
 - `-mg_coarse_pc_type`, `-mg_coarse_ksp_maxit`
- Each level k has prefix `mg_levels_ k _`
 - `-mg_levels_1_ksp_type`, `-mg_levels_2_pc_ilu_fill`
- Can automatically form coarse operators with the Galerkin process (`-pc_mg_galerkin`)
 - DMMG provides these automatically by interpolation

PETSc Extensibility

Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

EXTENDING CONFIGURE

- See `python/PETSc/packages/*.py` for examples
- Add module with class `Configure` derived from `config.base.Configure`
 - Can also derive from `PETSc.package.Package`
 - Implement `configure()` and `configureHelp()`
- Customize PETSc through the `make` system
 - `addDefine()`
 - `addTypedef()`, `addPrototype()`
 - `addMakeMacro()`, `addMakeRule()`
 - Deprecated `addSubstitution()`

LAYERING OVER PETSC

- SLEPc, TAO, and MagPar
 - Infrastructure and applications
- Use PETSc object structure
 - Dynamic dispatch
- Use dynamic linking facilities
 - Runtime type selection
- Use debugging and profiling tools
 - Memory management, runtime type checking

LINKING TO PETSC

- Nothing but the libraries
 - User can custom link
- Using only PETSc build variables
 - Include `bmake/common/variables`
- Also use PETSc build rules
 - Include `bmake/common/base`
- Also makes available 3rd party packages

ADDING AN IMPLEMENTATION

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

ADDING A NEW 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`
- Usually requires configure additions

ADDING A NEW 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) overridden methods
 - Construct any specific data
- Must also define a conversion to the base type
 - Only called in destructor right now

ADDING A NEW TYPE

- 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 NEW TYPE

- See `src/ksp/ksp/interface/dlregis.c`
- Define a package initializer (`PetscDLLLibraryRegister`)
 - 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 NEW TYPE

- 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 NEW TYPE

Things Swept Under The Rug

- Need `setFromOptions()` which allows implementation selection at runtime
- Have to manage the database of registered constructors
- View and destroy functions handled a little funny due to historical reasons

PETSc Future Plans

Matthew Knepley

PETSc Developer

Mathematics and Computer Science Division

Argonne National Laboratory

WHAT WILL PETSC BECOME?

PETSc has been, and will continue to be:

- a set of **libraries**,
- intended for solving **PDEs** in **parallel**,
- which present an **object-oriented** interface to the user.

WHAT IS NEW?

New classes and tools will be added to the existing PETSc framework:

- Unstructured mesh generation, refinement, and coarsening
- A high-level language for specification of weak forms (with **FFC**)
- Automatic generation of arbitrary finite element routines (with **FIAT**)
- Platform independent, **system** for configure, build, and distribution
- **Bindings** in multiple languages (C, F77, Python, Matlab, Mathematica)

PETSc PARTS

To make the new functionality easier for users to understand, PETSc will be divided conceptually into two parts:

- **PETSc-AS** will contain the components related to algebraic solvers, which is the current PETSc distribution.
- **PETSc-CS** will contain the support for continuum problems phrased in weak form. These modules will make use of the PETSc-AS modules in our implementation, but this is not strictly necessary.

However, the framework will remain integrated:

- Single release version, version control, and build system
- Each module is self-contained (install only what is necessary)

MESH CAPABILITIES

- 2D Delaunay generation and refinement
 - Triangle
- 3D Delaunay generation and refinement
 - TetGen
- 3D Delaunay generation, coarsening, and refinement
 - Allows quadratic Bezier elements
 - TUMBLE

MESH INCIDENCE

The interface is based upon a **fibre bundle model** which has a natural Grothendick topology. We have two equivalent ways of looking at a mesh, starting with the set of all mesh elements, meaning vertice, edges, faces, and cells. The first defines the mesh is a symmetric *incidence* relation over the elements:

$$\begin{aligned} e' \in \partial e & \quad \dim(e) > \dim(e') \\ eRe' \Leftrightarrow e \in \partial e' & \quad \dim(e) < \dim(e') \\ \partial e \cap \partial e' \neq \emptyset & \quad \dim(e') = \dim(e) \wedge \dim(\partial e \cap \partial e') = \dim(e) - 1 \end{aligned}$$

from which all connectivity information can be deduced. Combined with this we allow vector indexing by element

$$\text{VecSetValues}([e_1, e_5], [1.0, 0.0, 4.0, 3.0, 1.0, 0.0])$$

MESH COVERING

TO APPEAR SOON

Equivalently, we may define a topology based upon *covering*, which contains the same information as incidence. This formulation has the nice theoretical consequence, that the contravariant functor for element covering is field restriction, corresponding exactly to our element indexing.

A LANGUAGE FOR WEAK FORMS

In collaboration with

- Anders Logg of TTI
- Rob Kirby of University of Chicago

we have developed a small language for weak forms, based directly on an AST representation. The Fenics Form Compiler (FFC) processes each form algebraically, allowing some simplification and optimization, before passing it on to the integration generation routines.

INPUT LANGUAGE

We have a simple text language for input, incorporating:

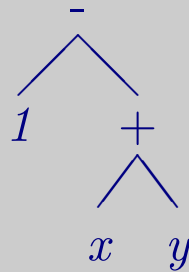
- Arithmetic, $+$, $-$, $*$, $/$, $\hat{}$, $()$ $\text{abs}(x)$
- Coordinate functions, $\cos(x)$ $\exp(x)$
- Continuum fields (known and unknown)
- Dual pairing, \langle , \rangle
- Matrix operations, $\text{TRANS}(\mathbf{u})$ $\text{DET}(\mathbf{u})$ $\text{VEC}(\mathbf{u})$
- Differential operators, $\text{GRAD}(\mathbf{u})$ $\text{DIV}(\mathbf{u})$ $\text{CURL}(\mathbf{u})$

WHAT IS AN AST?

A tree structure encoding the **syntactic** information in an expression. We may then associate **semantic** information with individual nodes or the tree itself. For example, our **Expression Trees** represent arithmetic expressions, so that:

- *Leaves* represent variables or constants
- *Internal* nodes represent operations on the leaves

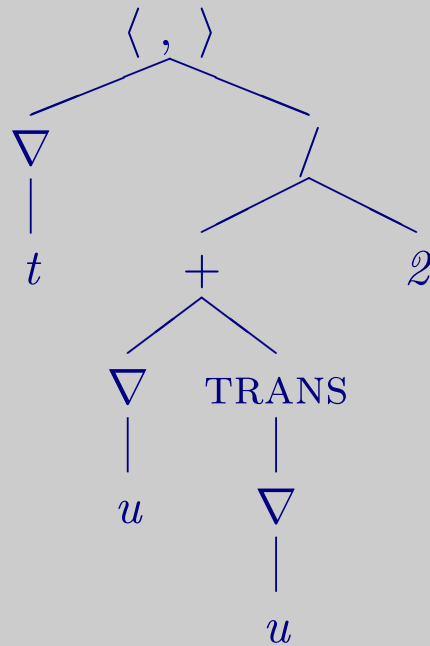
Here is an example tree representing $1 - x - y$



MORE GENERAL EXPRESSIONS

We allow more general expression than arithmetic in order to incorporate matrix algebra and weak form expressions

Here is another tree representing $\langle \nabla t, \frac{1}{2} (\nabla u + \nabla u^T) \rangle$



EXAMPLES

- Poisson Equation

$$\langle \text{grad } t, \text{grad } u \rangle - \langle t, 2*y*(2-y) + 2*x*(2-x) \rangle$$

- Vector Poisson Equation

$$\langle \text{grad vec } t, \text{grad vec } u \rangle - \langle \text{vec } t, \{4, 2*y*(2-y) + 2*x*(2-x)\} \rangle$$

- Linear Elasticity

$$\langle \text{grad vec } t, (\text{grad vec } u) + \text{trans } (\text{grad vec } u) \rangle - \langle \text{vec } t, \{6, 6\} \rangle$$

- Stokes Equation

$$\begin{aligned} \langle \text{grad } t, \text{grad } u \rangle - \langle t, \text{grad } p \rangle + \langle q, \text{div } u \rangle \\ - \langle \text{vec } t, \{4, -4\} \rangle + \langle q, 0 \rangle \end{aligned}$$

FIAT INTEGRATION

In order to generate a quadrature routines, we need:

- A differential form to integrate
- An element (usually a family and degree) using FIAT
- A quadrature rule

We then produce

- A C integration routines
- A Python module wrapper
- Optional Ferrari optimized routines
- Optional element assembly loop

BUILDSYSTEM

- Pure Python and **freely available**
- Currently handles configure and build
 - needs install and distribution
- Handles generated code through ASE
 - can uses md5, timestamp, diff, etc.
- Handles shared libraries for many architectures
 - Linux, Mac OSX, Windows

LANGUAGE BINDINGS

- Use the same code generation tools as the FE generation
 - ASE
- Each generator is just an AST Visitor
 - can be specified dynamically on the command line
- Custom bindings are easily handled
 - override methods from an existing generator
- New languages can be implemented dynamically
 - Uses SIDL model

With PETSc, we want less of this

With PETSc, we want less of this

```
dhx = (PetscReal)(info->mx-1); dhy = (PetscReal)(info->my-1);
```

```
hx = 1.0/dhx;                hy = 1.0/dhy;
```

```
hxdhy = hx*dhy;             hydhx = hy*dhx;
```

```
vx = x[j][i].u; avx = PetscAbsScalar(vx);
```

```
vxp = .5*(vx+avx); vxm = .5*(vx-avx);
```

```
vy = x[j][i].v; avy = PetscAbsScalar(vy);
```

```
vyp = .5*(vy+avy); vym = .5*(vy-avy);
```

```
f[j][i].omega = uxx + uyy +
```

```
  (vxp*(u - x[j][i-1].omega) +
```

```
   vxm*(x[j][i+1].omega - u)) * hy +
```

```
  (vyp*(u - x[j-1][i].omega) +
```

```
   vym*(x[j+1][i].omega - u)) * hx -
```

```
  0.5 * grashof * (x[j][i+1].temp - x[j][i-1].temp) * hy;
```

and more of this,

and more of this,

$$\frac{\partial \rho_f \phi}{\partial t} + \nabla \cdot [\rho_f \phi \mathbf{v}] = \Gamma \quad (1)$$

$$\frac{\partial \rho_s (1 - \phi)}{\partial t} + \nabla \cdot [\rho_s (1 - \phi) \mathbf{V}] = -\Gamma \quad (2)$$

$$-\frac{k_\phi}{\mu} [\nabla P - \rho_f \mathbf{g}] = \phi (\mathbf{v} - \mathbf{V}) \quad (3)$$

$$\nabla \cdot [\eta (\nabla \mathbf{V} + \nabla \mathbf{V}^T)] + \nabla [(\zeta - 2\eta/3) \nabla \cdot \mathbf{V}] + \bar{\rho} \mathbf{g} = \nabla P \quad (4)$$

$$\frac{T}{c_P} \left(\rho_s \alpha \bar{\mathbf{v}} \cdot \mathbf{g} - \frac{\Delta S}{\bar{\rho}} \Gamma \right) + \kappa \nabla^2 T + H = \frac{D_{\bar{\mathbf{v}}} T}{Dt} \quad (5)$$

so we can enjoy more of **this**.

so we can enjoy more of **this**.

