The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Computational and Applied Mathematics
Rice University

PETSc Tutorial
Rice Oil & Gas HPC Conference
Houston, TX Mar 2, 2016

Main Point

Never believe *anything*,

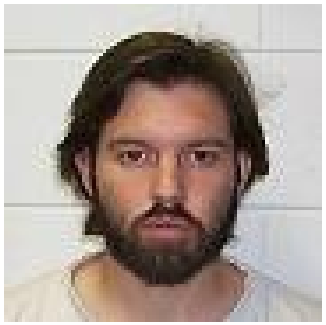
unless you can run it.

Main Point

Never believe *anything*,

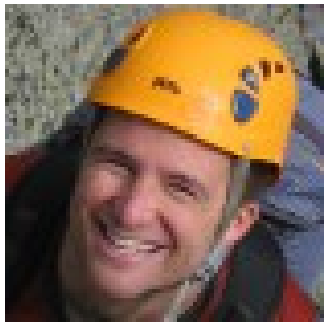
unless you can run it.

The PETSc Team


Matt Knepley


Barry Smith


Satish Balay


Jed Brown

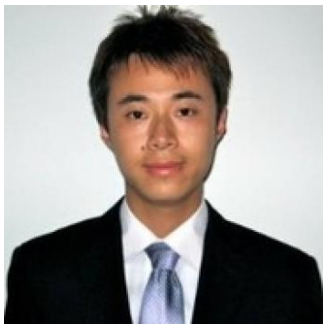

Hong Zhang

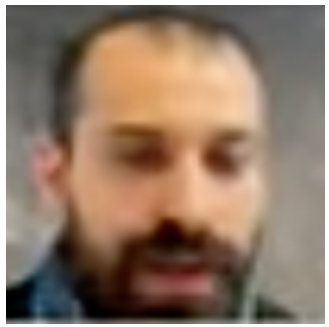

Lisandro Dalcin


Stefano Zampini


Mark Adams

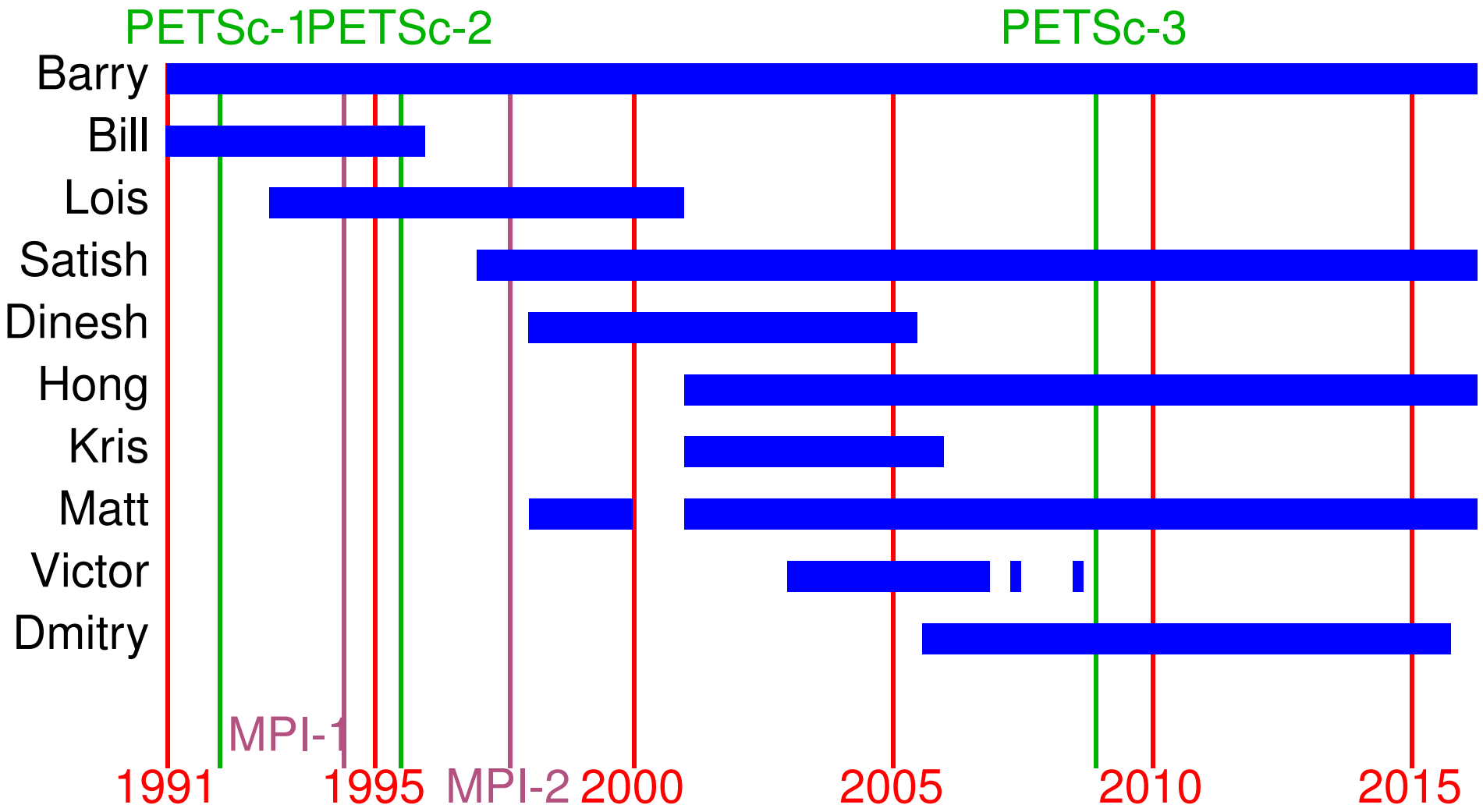

Toby Issac


Hong Zhang

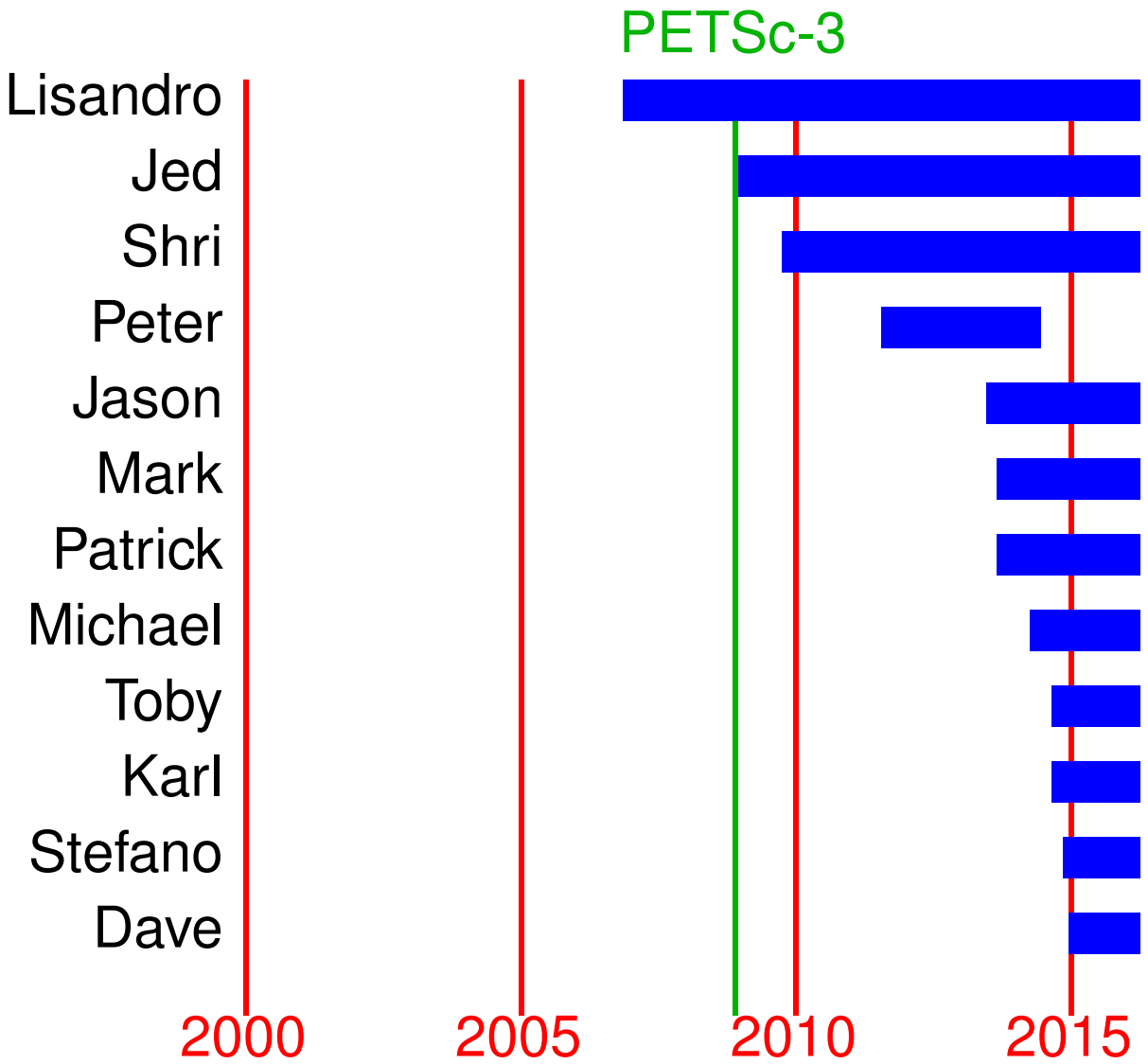

Pierre Jolivet


Junchao Zhang

Timeline (Old People)



Timeline (Young People)



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

Ask Questions!!!

- Helps **me** understand what you are missing
- Helps **you** clarify misunderstandings
- Helps **others** with the same question

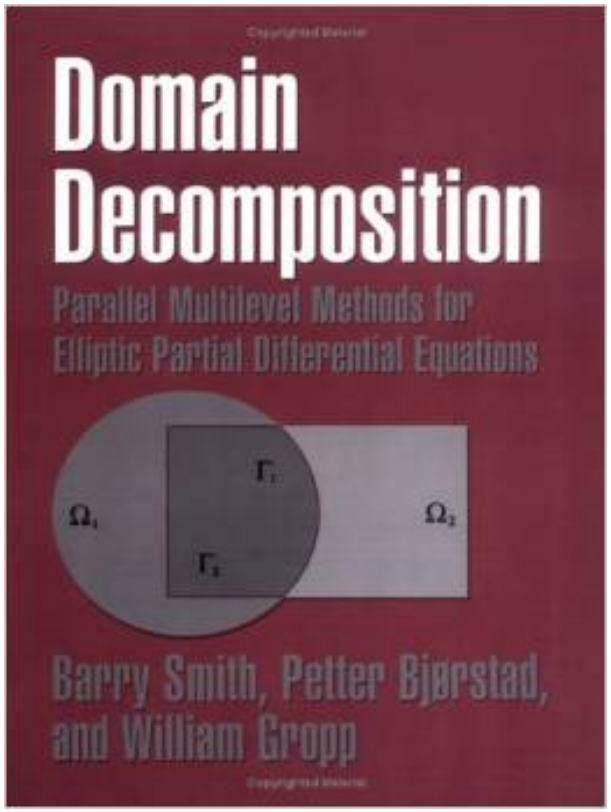
Outline

How did PETSc Originate?

PETSc was developed as a Platform for Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms
  - which blur these boundaries



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.

— Barry Smith

Advice from Bill Gropp

You want to think about how you decompose your data structures, how you think about them globally. [...] If you were building a house, you'd start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn't start with a bunch of tiles and say. "Well I'll put this tile down on the ground, and then I'll find a tile to go next to it." But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you're going to survive making your code parallel.

(http://www.rce-cast.com/Podcast/rce-28-mpich2.html)

What is PETSc?

A freely available and supported research code for the parallel solution of nonlinear algebraic equations

Free

- Download from http://www.petsc.org
- Free for everyone, including industrial users

Supported

- 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, Matlab, Julia, and Python

What is PETSc?

- Portable to any parallel system supporting MPI, including:
  - Tightly coupled systems
    - Cray XT6, BG/Q, NVIDIA Fermi, K Computer
  - Loosely coupled systems, such as networks of workstations
    - IBM, Mac, iPad/iPhone, PCs running Linux or Windows
- PETSc History
  - Begun September 1991
  - Over 60,000 downloads since 1995 (version 2)
  - Currently 400 per month
- PETSc Funding and Support
  - Department of Energy
    - ECP, PSAAPIII, AMR, BES, SciDAC, MICS
  - National Science Foundation
    - CSSI, SI2, CIG, CISE
  - Intel Parallel Computing Center

Outline

Who Uses PETSc?

Computational Scientists

- Earth Science
  - PyLith (CIG)
  - Underworld (Monash)
  - Salvus (ETHZ)
  - TerraFERMA (LDEO, Columbia, Oxford)
- Multiphysics
  - MOOSE
  - GRINS
- Subsurface Flow and Porous Media
  - PFLOTRAN (DOE)
  - STOMP (DOE)

Who Uses PETSc?

Computational Scientists

- CFD
  - IBAMR
  - Fluidity
  - OpenFVM
- Fusion
  - XGC
  - BOUT++
  - NIMROD
  - $M3D - C^1$

Who Uses PETSc?

Algorithm Developers

- Iterative methods
  - Deflated GMRES
  - LGMRES
  - QCG
  - SpecEst
- Preconditioning researchers
  - FETI-DP (Klawonn and Rheinbach)
  - STRUMPACK (Ghysels and Li)
  - HPDDM (Jolivet and Nataf)
  - ParPre (Eijkhout)

Who Uses PETSc?

Algorithm Developers

- Discretization
  - Firedrake
  - FEniCS
  - libMesh
  - Deal II
  - PETSc-FEM
  - OOFEM
  - PetRBF
- Outer Loop Solvers
  - Eigensolvers (SLEPc)
  - Optimization (PERMON)

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?

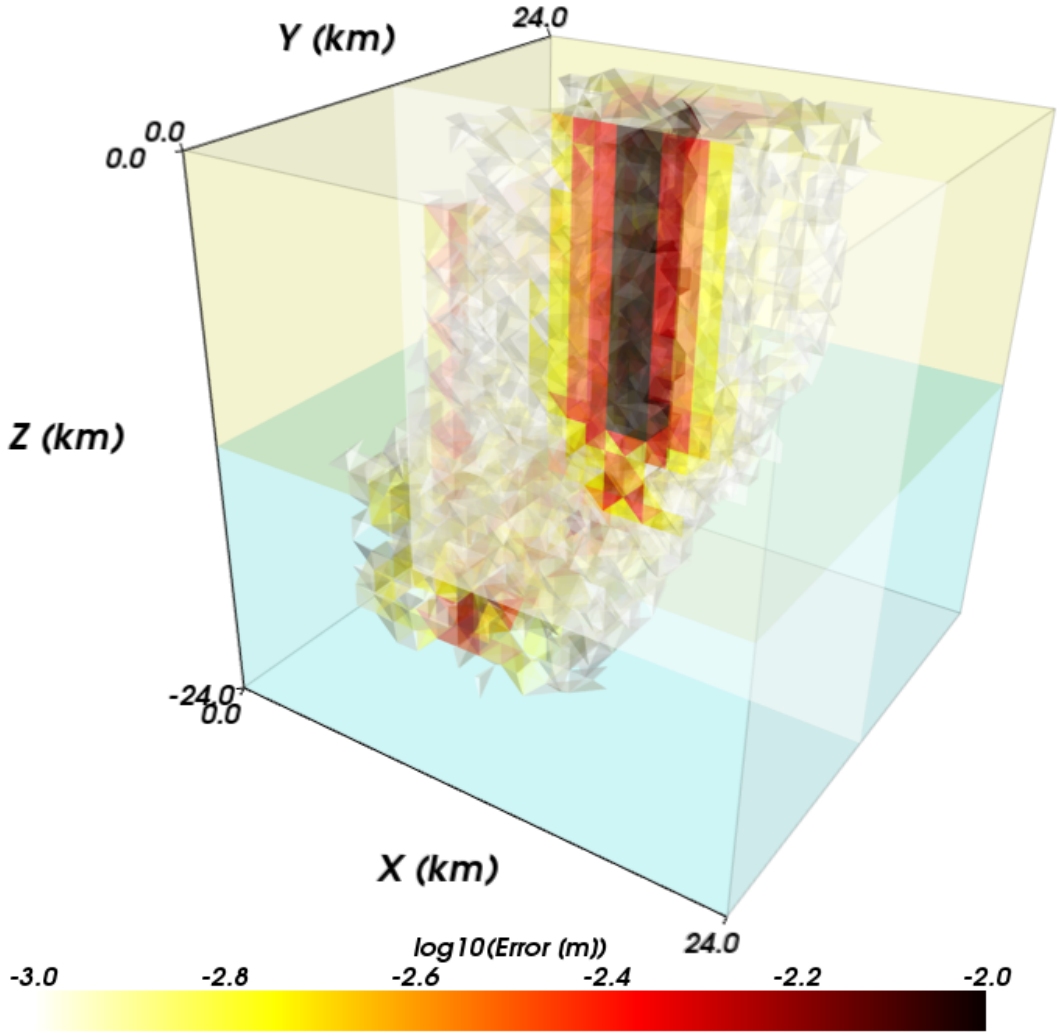
- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media
- PETSc has run on over 1,500,000 cores efficiently
  - Gordon Bell Prize Mantle Convection on IBM BG/Q Sequoia
- PETSc applications have run at 23% of peak (600 Teraflops)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

PyLith

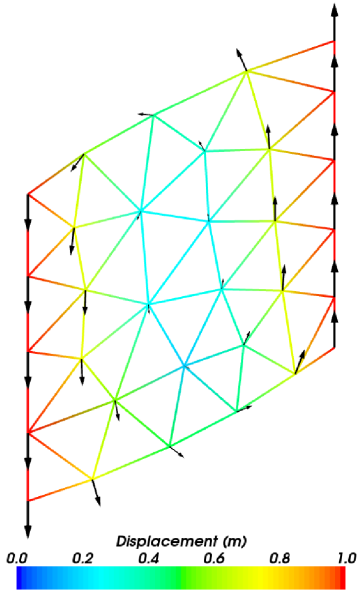
- Multiple problems
  - Dynamic rupture
  - Quasi-static relaxation
  - Numerical Green functions
  - Earthquake cycle
- Multiple models
  - Nonlinear visco-plastic
  - Finite deformation
  - Fault constitutive models
- Multiple meshes
  - 1D, 2D, 3D
  - Hex and tet meshes
  - Refinement and fault insertion
- Scalable, parallel solvers

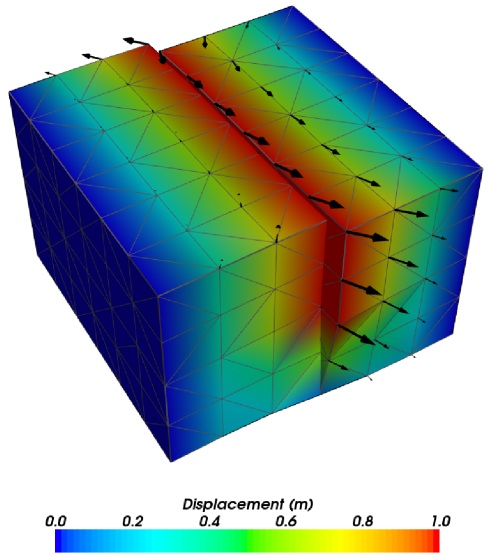


[a] Aagaard, Knepley, Williams

Multiple Mesh Types

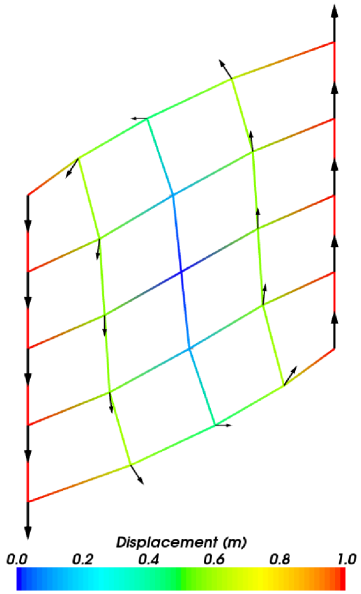
Triangular

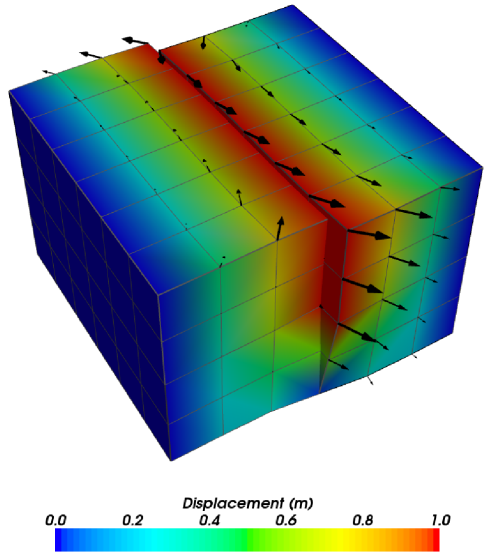




Tetrahedral

Rectangular

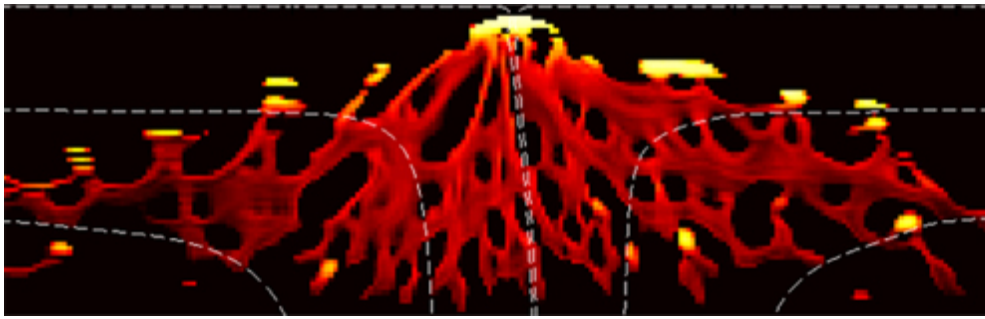


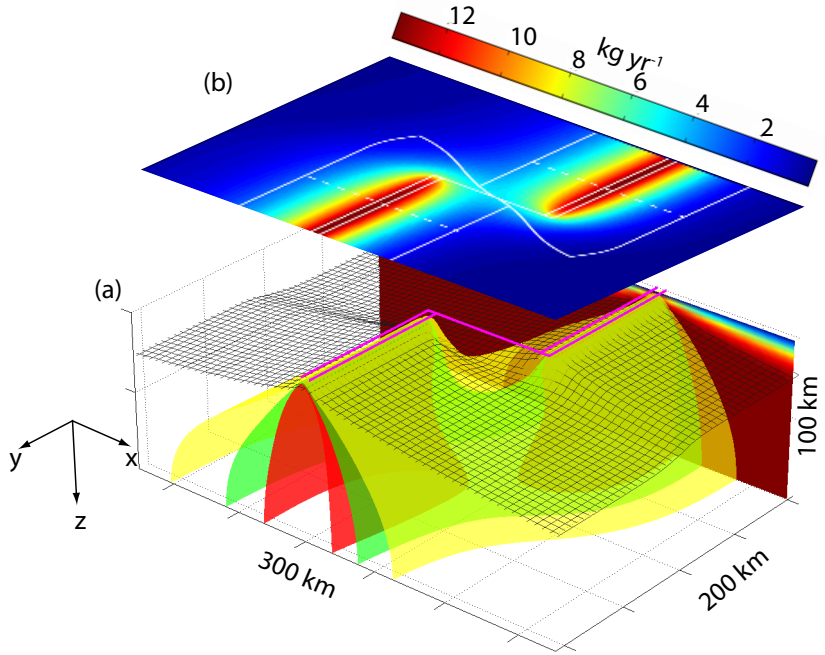


Hexahedral

Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

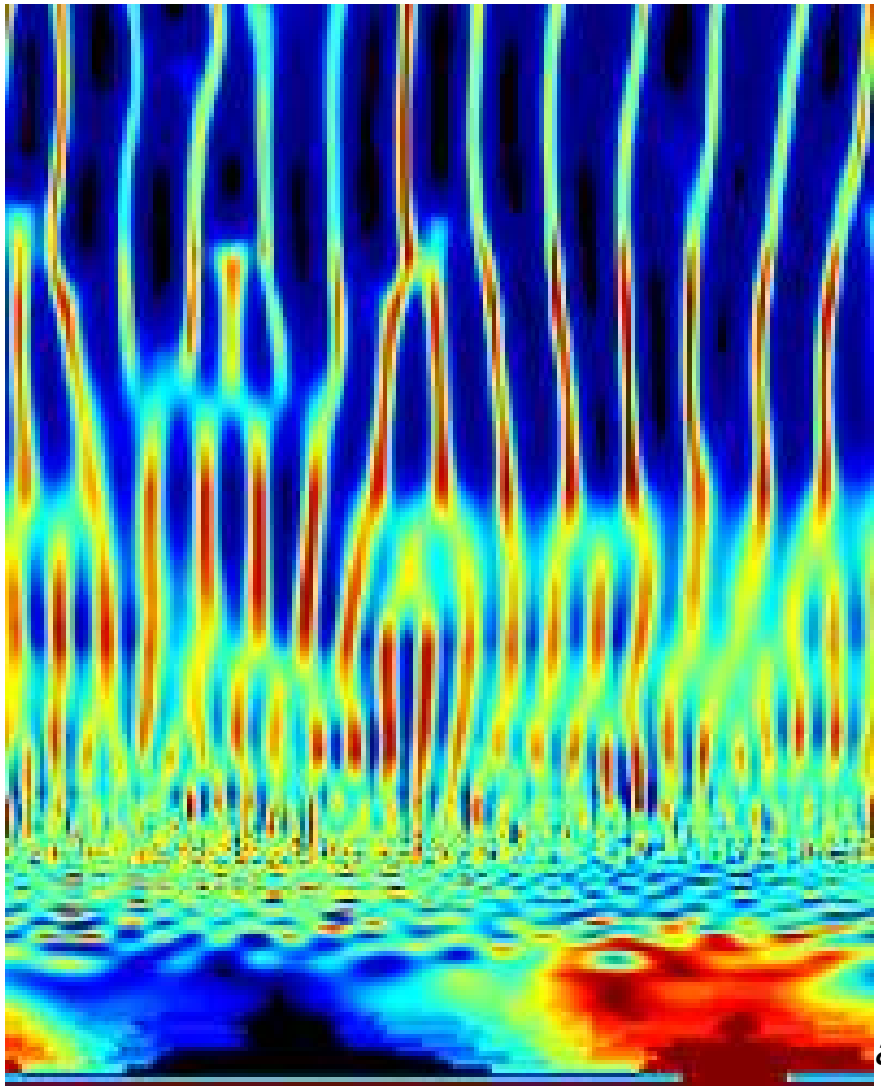




[a]Katz

Magma Dynamics

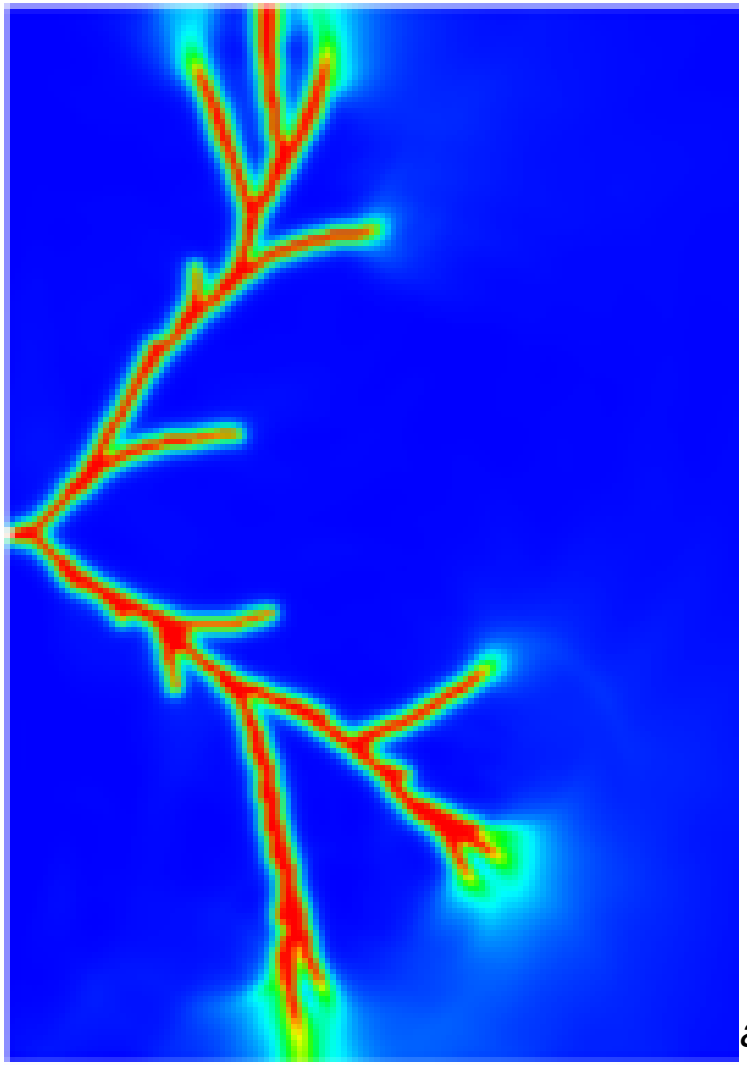
- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit



[a]Katz, Speigelman

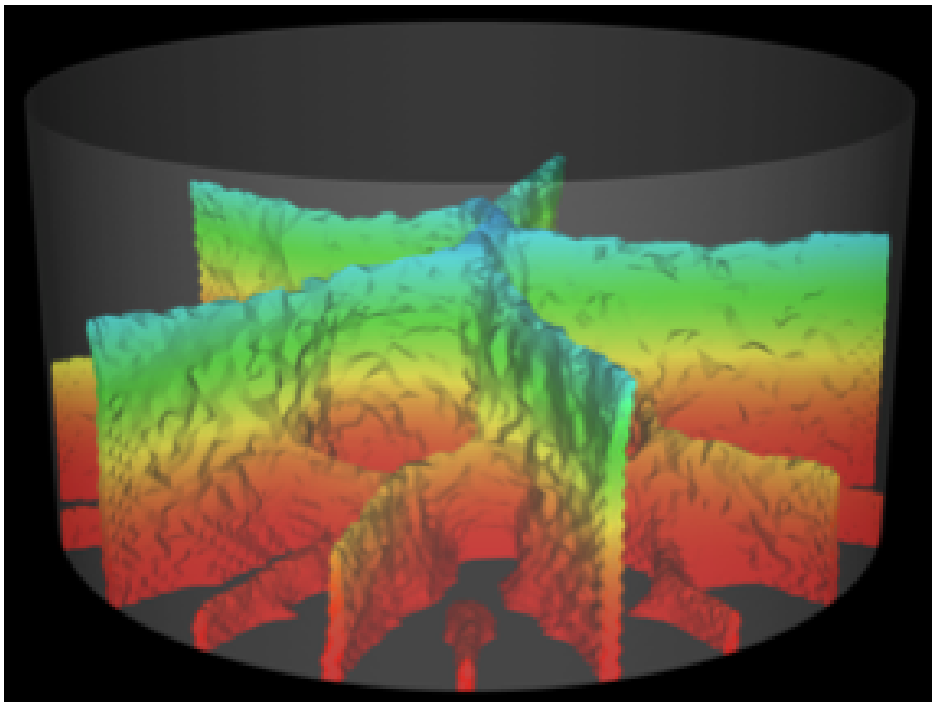
Fracture Mechanics

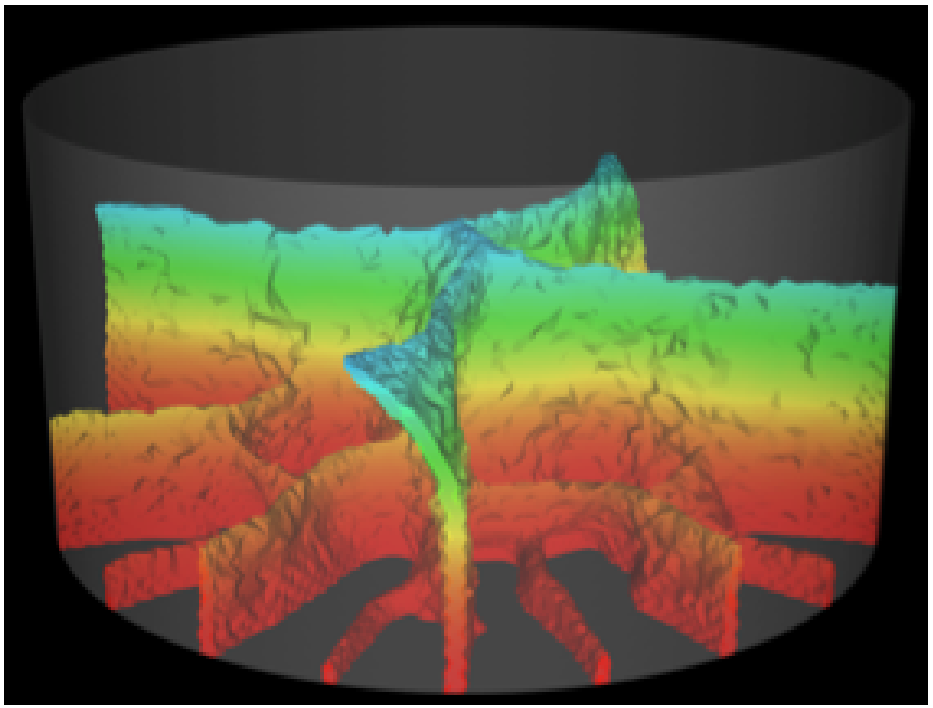
- Full variational formulation
  - Phase field
  - Linear or Quadratic penalty
- Uses TAO optimization
  - Necessary for linear penalty
  - Backtacking
- No prescribed cracks (movie)
  - Arbitrary crack geometry
  - Arbitrary intersections
- Multiple materials
  - Composite toughness



[a]Bourdin

Fracture Mechanics





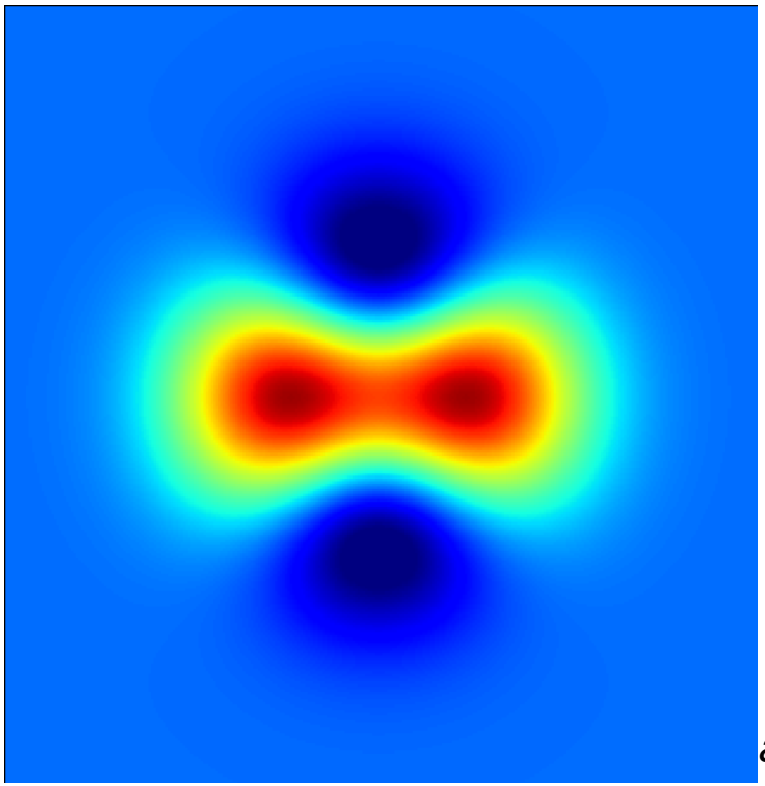
[1]

[1] Bourdin

Vortex Method

t = 000

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

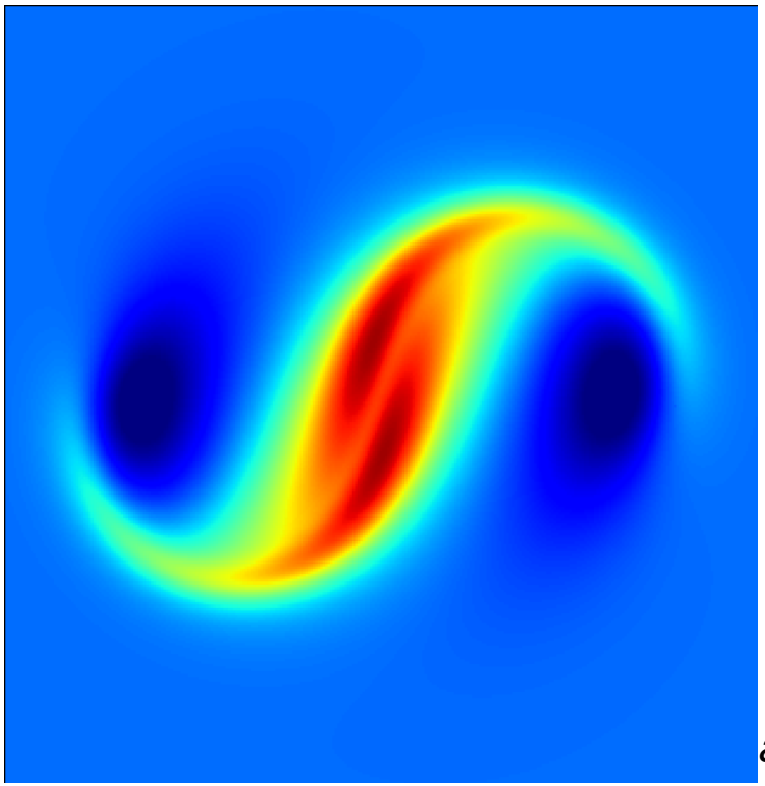


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 100

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

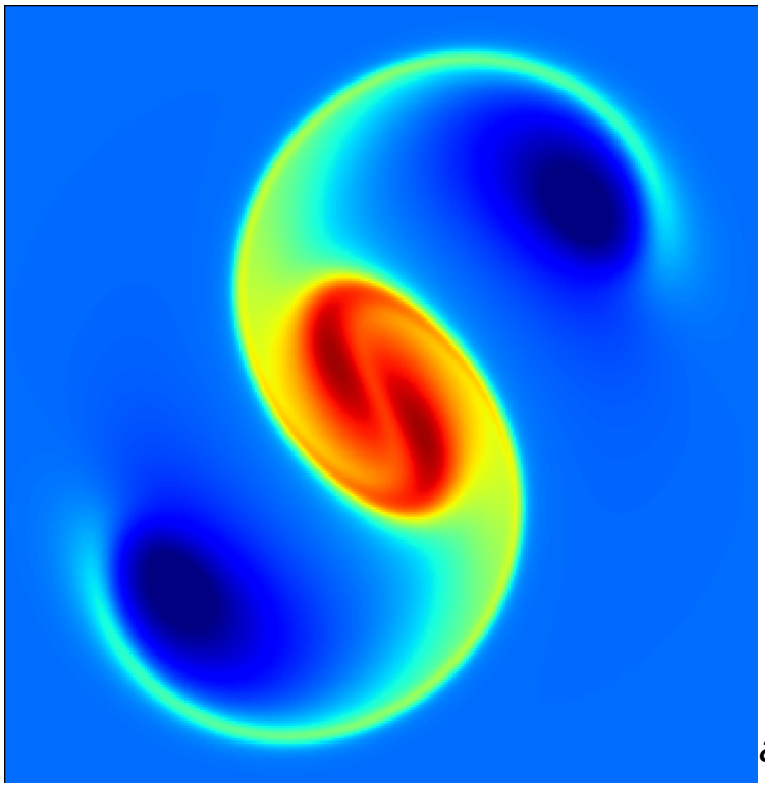


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 200

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

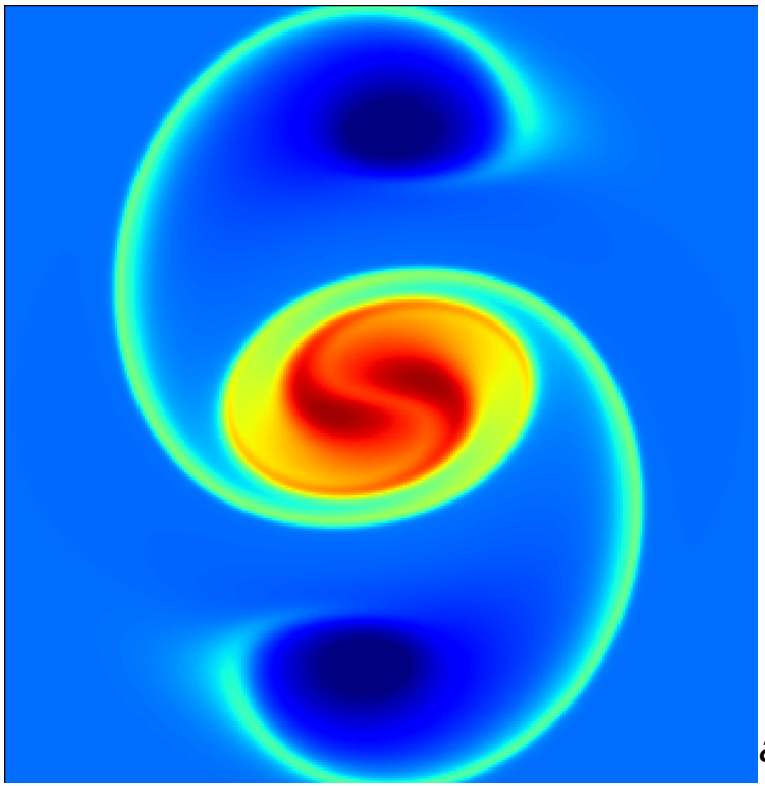


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 300

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

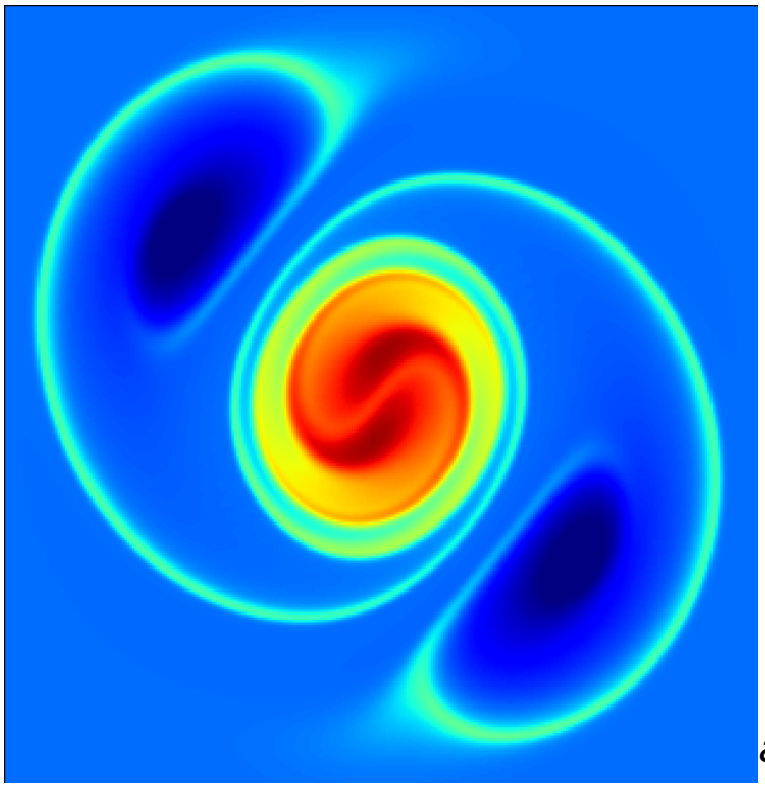


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 400

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

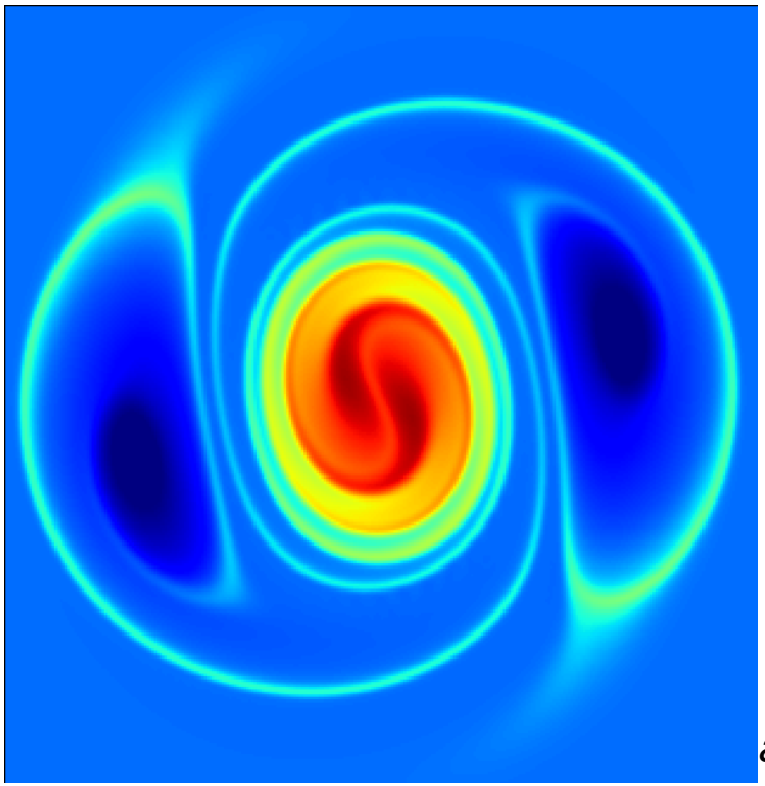


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 500

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

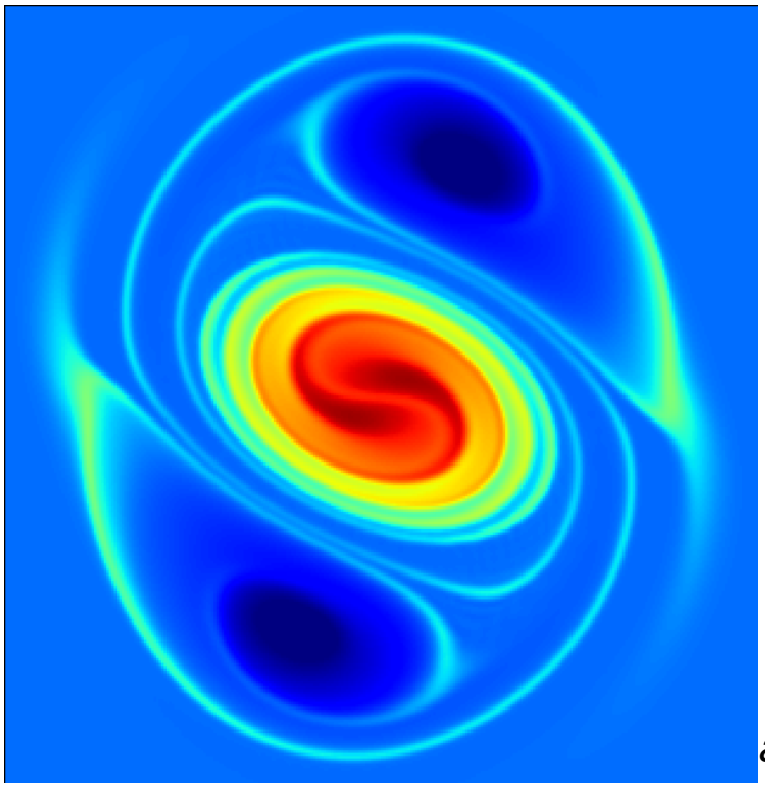


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 600

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

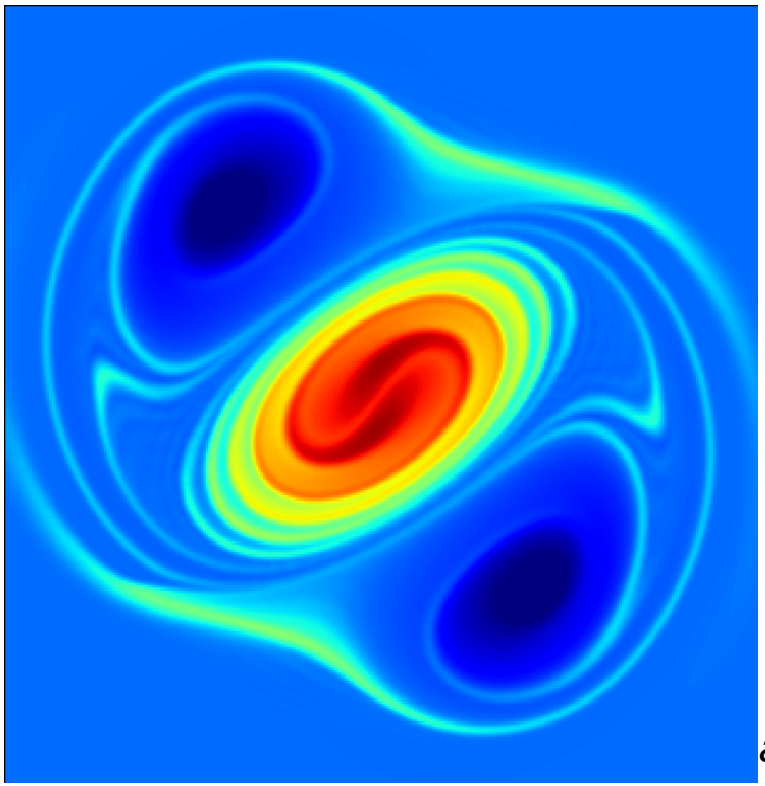


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 700

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

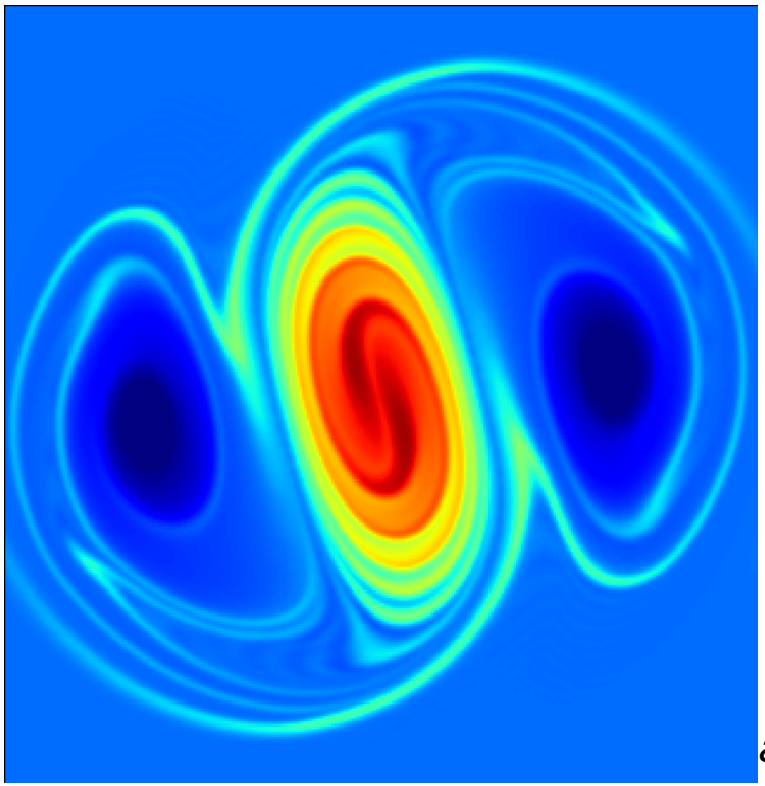


[a]Cruz, Yokota, Barba, Knepley

Vortex Method

t = 800

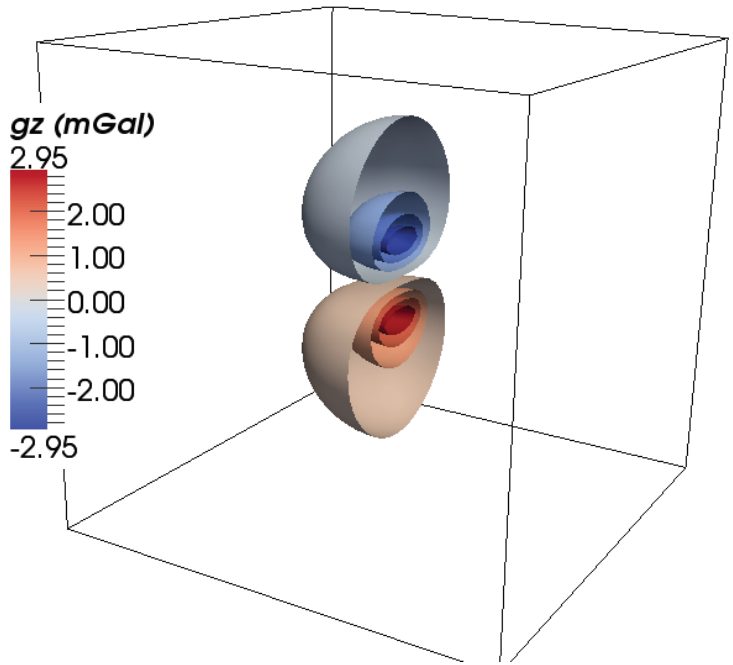
- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU



[a]Cruz, Yokota, Barba, Knepley

Gravity Anomaly Modeling

- Potential Solution
  - Kernel of inverse problem
  - Needs optimal algorithm
- Implementations
  - Direct Summation
  - FEM
  - FMM
- Parallelism
  - MPI
  - 4000+ cores
  - All methods scalable



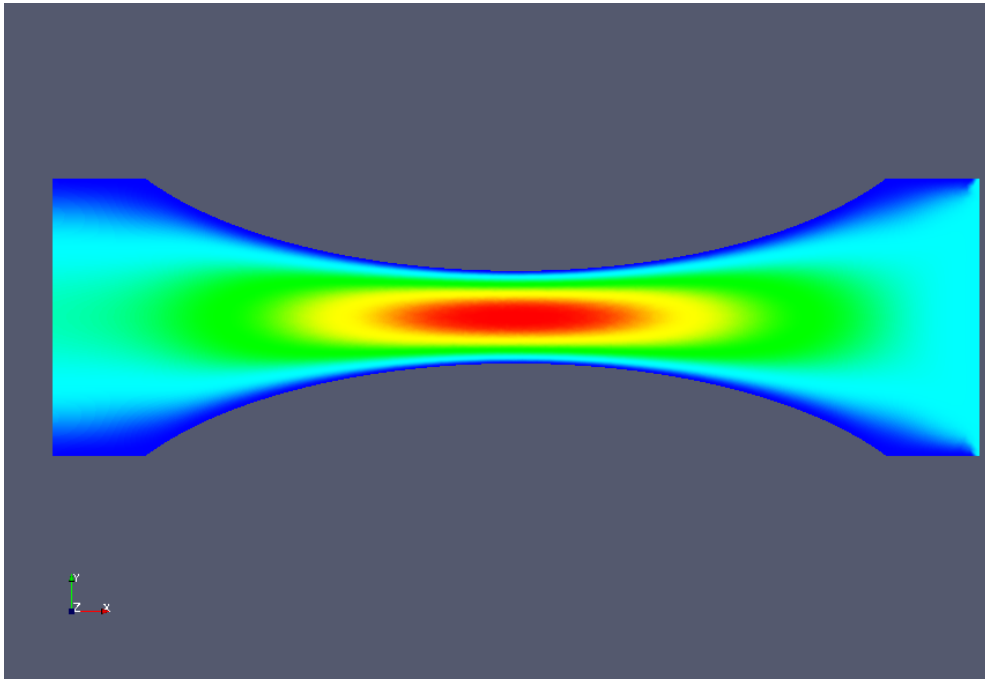
a

[a] May, Knepley

FEniCS-Apps

Rheagen

- Rheologies
  - Maxwell
  - Grade 2
  - Oldroyd-B
- Stabilization
  - DG
  - SUPG
  - EVSS
  - DEVSS
  - Macroelement
- Automation
  - FIAT (elements)
  - FFC (weak forms)

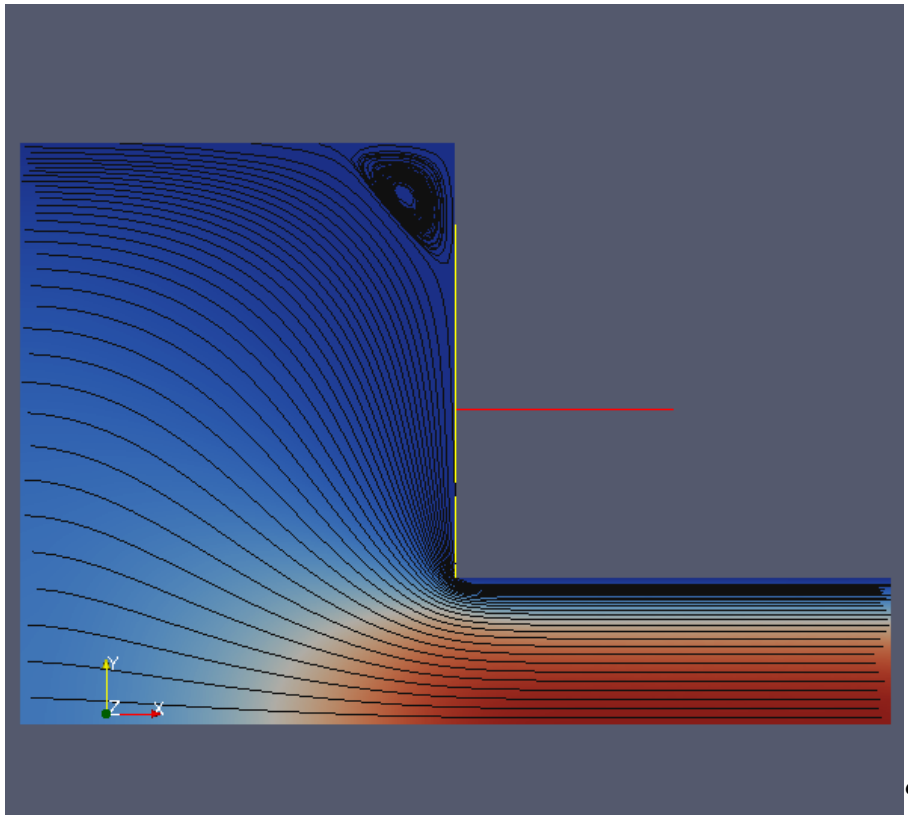


[a] Terrel

FEniCS-Apps

Rheagen

- Rheologies
  - Maxwell
  - Grade 2
  - Oldroyd-B
- Stabilization
  - DG
  - SUPG
  - EVSS
  - DEVSS
  - Macroelement
- Automation
  - FIAT (elements)
  - FFC (weak forms)

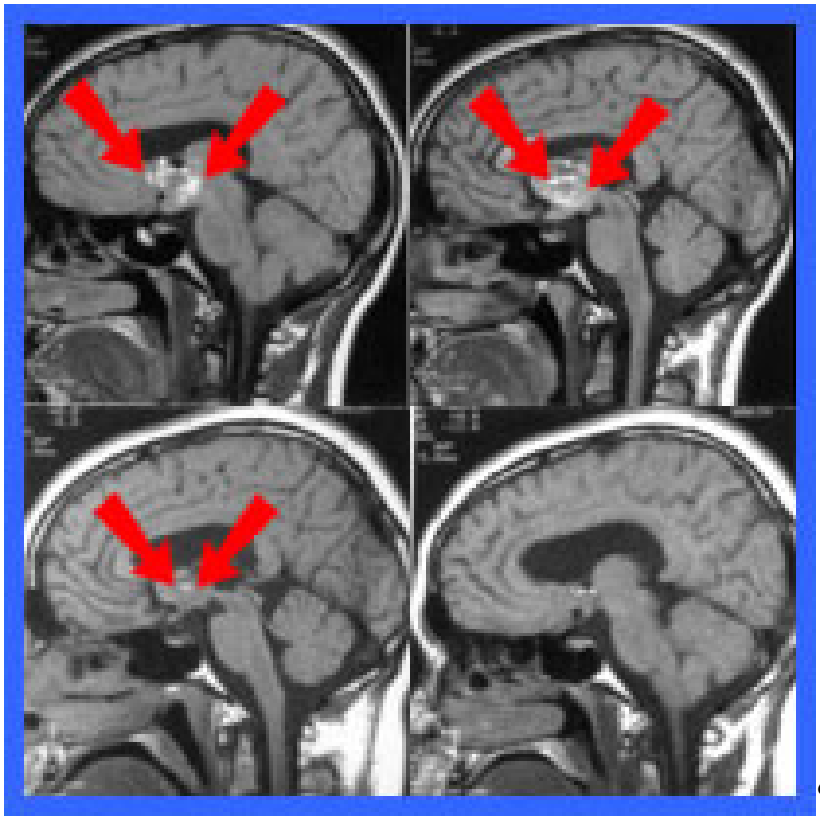


a

[a] Terrel

Real-time Surgery

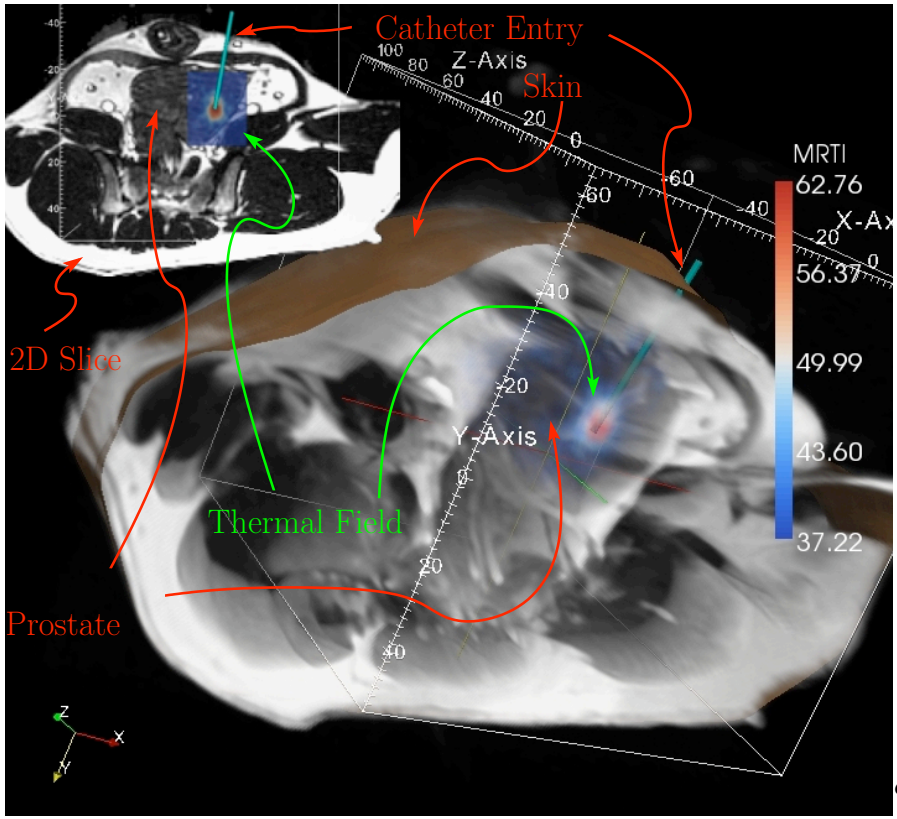
- Brain Surgery
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon
- Laser Thermal Therapy
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem



[a]Warfield, Ferrant, et.al.

Real-time Surgery

- Brain Surgery
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon
- Laser Thermal Therapy
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem



[a]

[a] Fuentes, Oden, et.al.

Outline

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

Outline

Downloading PETSc

- There is a Git repository
- The latest tarball is on the PETSc site: https://ftp.mcs.anl.gov/pub/petsc/release-snapshots/
- There is a pip package (`pip install petsc petsc4py`)
- There is a Debian package (`aptitude install petsc-dev`)

Cloning PETSc

- The full development repository is open to the public
  - https://gitlab.com/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 v3.18.0

Unpacking PETSc

- Just clone development repository
  - `git clone http://gitlab.com/petsc/petsc.git`
  - `git checkout -rv3.18.0`

or

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

Exercise 1

Download and Unpack PETSc!

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

Configuring PETSc for FEM

$PETSC_DIR/configure

–download-triangle –download-ctetgen
–download-p4est –download-egads
–download-eigen –download-pragmatic
–download-metis –download-parmetis
–download-hdf5 –download-netcdf
–download-pnetcdf –download-exodusii
–download-ml –download-superlu –download-superlu_dist
–download-scalapack –download-mumps
–download-fftw –download-slepc –download-bamg

Configuring PETSc for FEM

$PETSC_DIR/configure

–download-triangle –download-ctetgen
–download-p4est –download-egads

–download-eigen –download-pragmatic
–download-metis –download-parmetis

–download-hdf5 –download-netcdf
–download-pnetcdf –download-exodusii

–download-ml –download-superlu –download-superlu_dist
–download-scalapack –download-mumps

–download-fftw –download-slepc –download-bamg

Configuring PETSc for FEM

$PETSC_DIR/configure

–download-triangle –download-ctetgen
–download-p4est –download-egads

–download-eigen –download-pragmatic
–download-metis –download-parmetis

–download-hdf5 –download-netcdf
–download-pnetcdf –download-exodusii

–download-ml –download-superlu –download-superlu_dist
–download-scalapack –download-mumps

–download-fftw –download-slepc –download-bamg

Configuring PETSc for FEM

$PETSC_DIR/configure

–download-triangle –download-ctetgen
–download-p4est –download-egads

–download-eigen –download-pragmatic
–download-metis –download-parmetis

–download-hdf5 –download-netcdf
–download-pnetcdf –download-exodusii

–download-ml –download-superlu –download-superlu_dist
–download-scalapack –download-mumps

–download-fftw –download-slepc –download-bamg

Configuring PETSc for FEM

$PETSC_DIR/configure

–download-triangle –download-ctetgen
–download-p4est –download-egads

–download-eigen –download-pragmatic
–download-metis –download-parmetis

–download-hdf5 –download-netcdf
–download-pnetcdf –download-exodusii

–download-ml –download-superlu –download-superlu_dist
–download-scalapack –download-mumps

–download-fftw –download-slepc –download-bamg

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
  - GMP, MPFR
  - ConcurrencyKit, hwloc
  - MPICH, OpenMPI
  - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
  - SuiteSparse, MUMPS, SuperLU, SuperLU_Dist, PaStiX, Pardiso
  - SLEPc, HYPRE, ML
  - BLOPEX, FFTW, STRUMPACK, SPAI, CUSP, Sundials
  - Triangle, TetGen, p4est, Pragmatic
  - HDF5, NetCDF, ExodusII
  - AfterImage, gifLib, libjpeg, opengl

Exercise 2

Configure your downloaded PETSc.

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

Exercise 3

Build your configured PETSc.

Exercise 4

Reconfigure PETSc to use ParMetis.

1. `linux-debug/lib/petsc/conf/reconfigure-linux-debug.py`
   - `--PETSC_ARCH=arch-linux-parmetis`
   - `--download-metis --download-parmetis`
2. `PETSC_ARCH=linux-parmetis make`
3. `PETSC_ARCH=linux-parmetis make check`

Outline

Running PETSc

- Try running PETSc examples first
  - `cd $PETSC_DIR/src/snes/tutorials`
- Build examples using make targets
  - `make ex5`
- Run using MPI directly
  - `./ex5 -snes_max_it 5`
  - `mpirun -np 2 ./ex5 -snes_max_it 5`
  - `mpiexec -n 2 ./ex5 -snes_monitor`

Running PETSc with Python

- Can run any PETSc example
  - `./config/builder2.py check ./src/snes/examples/tutorials/ex5.c`
- Checks against test output
  - Ignores if no output is present
- Can specify multiple files
  - `builder2.py check [./src/snes/examples/tutorials/ex5.c,code.c]`
- Can also run using MPI directly
  - Use `--retain` to keep executable
  - `mpiexec ./$PETSC_ARCH/lib/lib-ex5/ex5 -snes_monitor`

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())
- Can remap PETSc objects to smaller communicators

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`
- Generic viewing option
  - `-foo_view <type>:<filename>:<format>:<file mode>`

Common Monitoring Options

- Display the residual
  - `-ksp_monitor`
- 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.petsc.org
- Hyperlinked documentation
  - Online 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

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
- Call PetscLogStagePush() and PetscLogStagePop()
  - User can add new stages
- Call PetscLogEventBegin() and PetscLogEventEnd()
  - User can add new events

Command Line Processing

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

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_Commcomm, Mat*A)
- MatSetSizes(MatA, PetscInt m, PetscInt n, PetscInt M, PetscInt N)
- MatSetType(MatA, MatType typeName)
- MatSetFromOptions(MatA)
  - Can set the type at runtime
- MatSeqAIJPreallocation(MatA, PetscIntnz, const PetscInt nnz[])
- MatXAIJPreallocation(MatA, bs, dnz[], onz[], dnzu[], onzu[])
- MatSetValues(MatA, 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

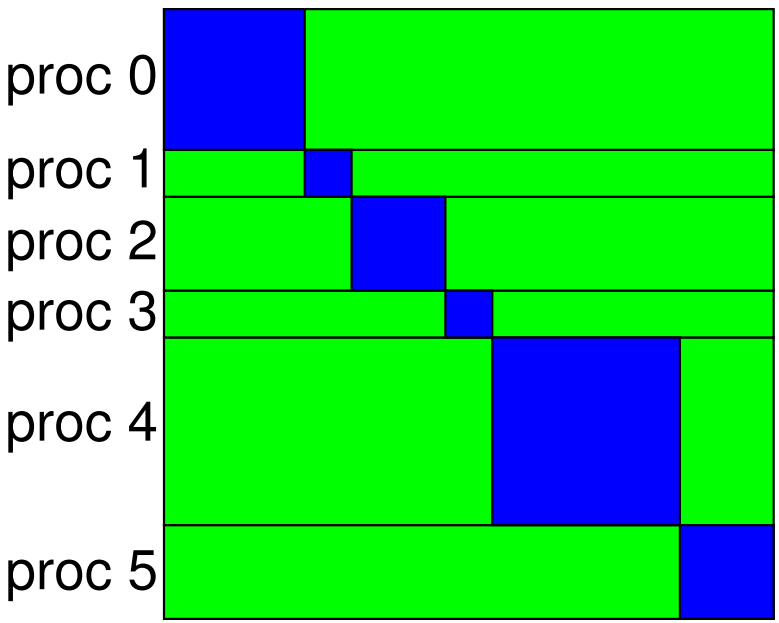
- A three step process
  - Each process sets or adds values
  - Begin communication to send values to the correct process
  - Complete the communication
- MatSetValues(A, m, rows[], n, cols [], values [], mode)
  - mode is either INSERT_VALUES or ADD_VALUES
  - Logically dense block of values
- Two phase 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

Programming Languages for Scientific Computing

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

```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

```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

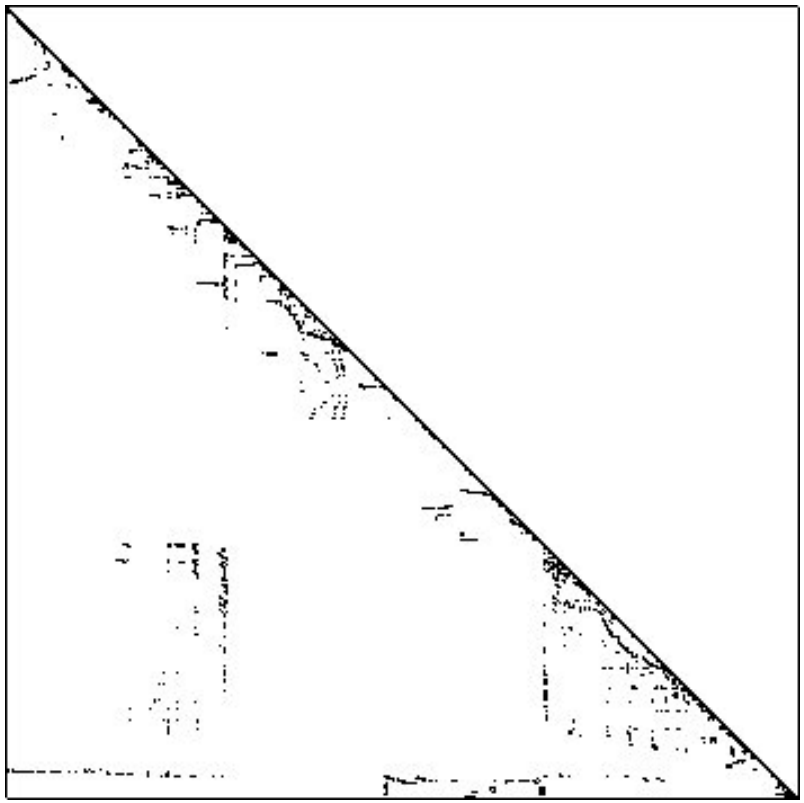
Matrix Memory Preallocation

Sequential Sparse Matrices

MatSeqAIJPreallocation(MatA, int nz, int nnz[])

nz: expected number of nonzeros in any row

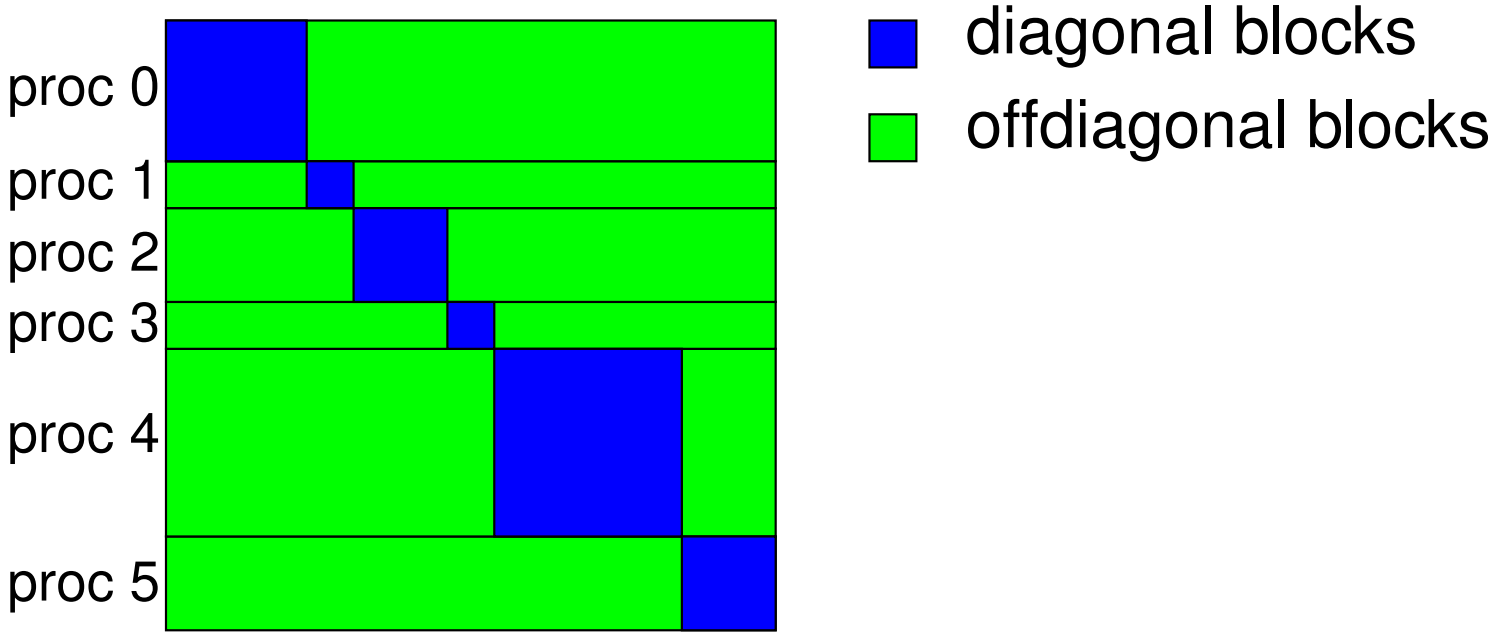
nnz(i): expected number of nonzeros in row i



Matrix Memory Preallocation

ParallelSparseMatrix

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



- 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(MatA, 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()`

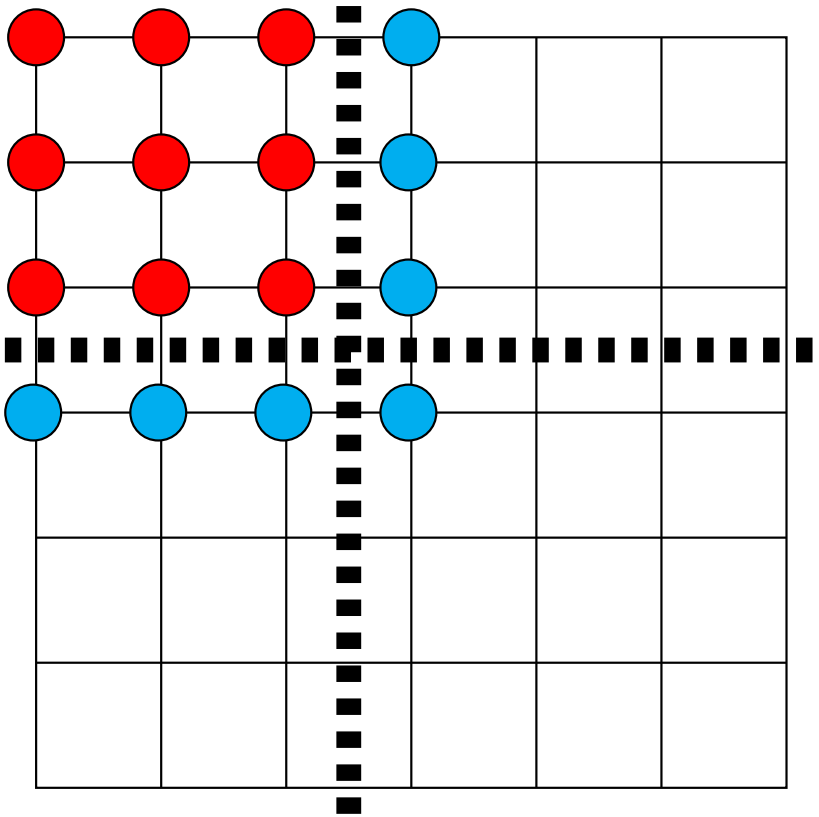
When PETSc needs to evaluate the nonlinear residual **F(x)**,

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

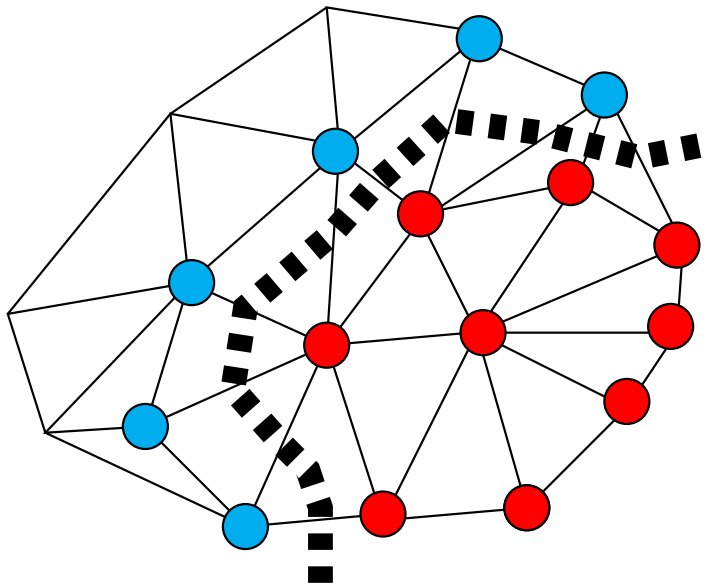
Ghost Values

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



- Local Node
- Ghost Node



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

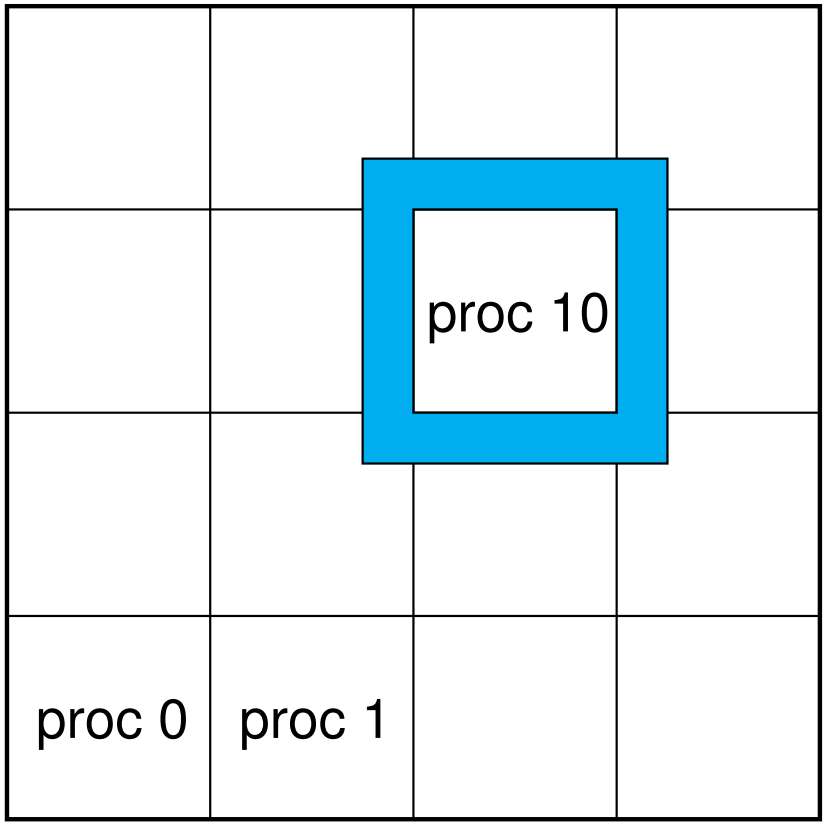
Two-step process enables overlapping computation and communication

- 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

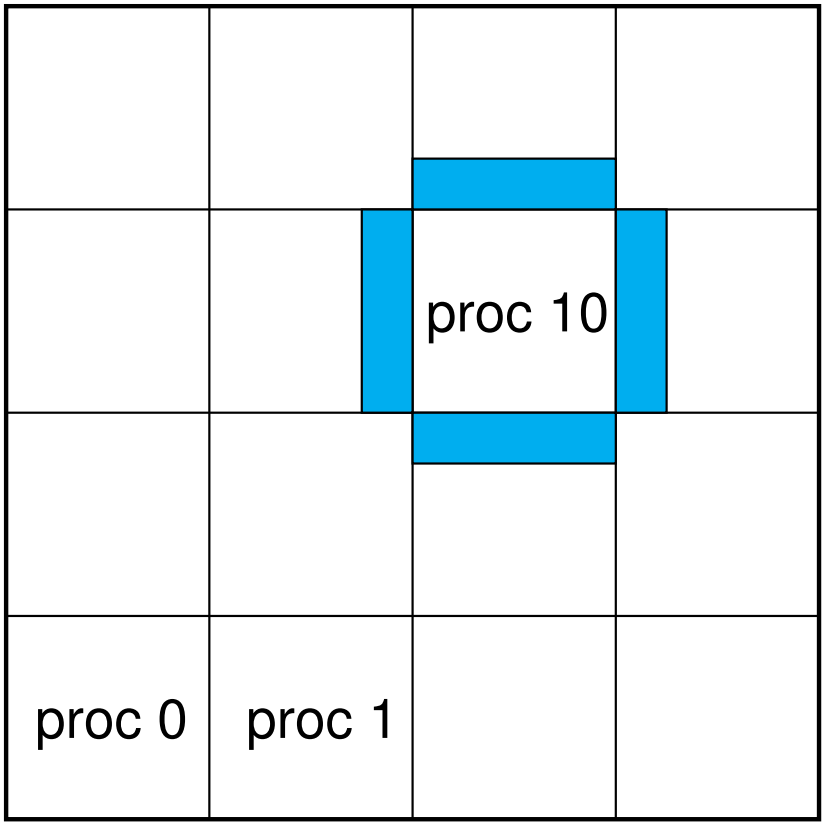
The process can be reversed with DALocalToGlobalBegin/End().

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`

Outline

The Great Solver Schism: Monolithic or Split?

Monolithic	Split
Direct solvers	Physics-split Schwarz (based on relaxation)
Coupled Schwarz	Physics-split Schur (based on factorization)
Coupled Neumann-Neumann (use unassembled matrices)	SIMPLE, PCD, LSC
Coupled Multigrid	segregated smoothers
	Augmented Lagrangian

Need to understand

Local spectral properties	Global coupling strengths
Compatibility properties	

Preferred data structures depend on which method is used.

Outline

FieldSplit Preconditioner

- Analysis
  - Use **IS**es to define fields
  - Decouples **PC** from problem definition
- Synthesis
  - Additive, Multiplicative, Schur
  - Commutes with Multigrid

FieldSplit Customization

- Analysis
  - `-pc_fieldsplit_<split num>_fields 2,1,5`
  - `-pc_fieldsplit_detect_saddle_point`
- Synthesis
  - `-pc_fieldsplit_type <additive, multiplicative, schur>`
  - `-pc_fieldsplit_diag_use_amat`
    `-pc_fieldsplit_off_diag_use_amat`
    Use diagonal blocks of operator to build PC
- Schur complements
  - `-pc_fieldsplit_schur_precondition <user,a11,full,self,selfp>`
    How to build preconditioner for S
  - `-pc_fieldsplit_schur_factorization_type <diag,lower,upper,full>`
    Which off-diagonal parts of the block factorization to use

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Block-Jacobi (Exact), Cohouet & Chabard, IJNMF, 1988.

```
-ksp_type gmres -pc_type fieldsplit -pc_fieldsplit_type additive
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Block-Jacobi (Inexact), Cohouet & Chabard, IJNMF, 1988.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type additive
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Gauss-Seidel (Inexact), Elman, DTIC, 1994.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type multiplicative
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Gauss-Seidel (Inexact), Elman, DTIC, 1994.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type multiplicative
 -pc_fieldsplit_0_fields 1 -pc_fieldsplit_1_fields 0
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type preonly -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} I & B^T \\ 0 & \hat{A} \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Diagonal Schur Complement, Olshanskii, et.al., Numer. Math., 2006.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type diag
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type minres  -fieldsplit_pressure_pc_type none
```

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Lower Schur Complement, May and Moresi, PEPI, 2008.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type lower
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type minres  -fieldsplit_pressure_pc_type none
```

$$\begin{pmatrix} \hat{A} & 0 \\ B^T & \hat{S} \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Upper Schur Complement, May and Moresi, PEPI, 2008.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type upper
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type gamg
 -fieldsplit_pressure_ksp_type minres  -fieldsplit_pressure_pc_type none
```

$$\begin{pmatrix} \hat{A} & B \\ & \hat{S} \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Uzawa Iteration, Uzawa, 1958

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type upper
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_type richardson -fieldsplit_pressure_pc_type jac
 -fieldsplit_pressure_ksp_max_it 1
```

$$\begin{pmatrix} A & B \\ & \hat{S} \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Full Schur Complement, Schur, 1905.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_rtol 1e-10 -fieldsplit_pressure_pc_type jacobi
```

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

SIMPLE, Patankar and Spalding, IJHMT, 1972.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
 -fieldsplit_velocity_ksp_type preonly -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_rtol 1e-10 -fieldsplit_pressure_pc_type jacobi
 -fieldsplit_pressure_inner_ksp_type preonly
  -fieldsplit_pressure_inner_pc_type jacobi
 -fieldsplit_pressure_upper_ksp_type preonly
  -fieldsplit_pressure_upper_pc_type jacobi
```

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & B^T D_A^{-1} B \end{pmatrix} \begin{pmatrix} I & D_A^{-1} B \\ 0 & I \end{pmatrix}$$

Solver Configuration: No New Code

ex62: P_2/P_1 Stokes Problem on Unstructured Mesh

Least-Squares Commutator, Kay, Loghin and Wathen, SISC, 2002.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_schur_factorization_type full
-pc_fieldsplit_schur_precondition self
 -fieldsplit_velocity_ksp_type gmres -fieldsplit_velocity_pc_type lu
 -fieldsplit_pressure_ksp_rtol 1e-5 -fieldsplit_pressure_pc_type lsc
```

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & \hat{S}_{\text{LSC}} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

Solver Configuration: No New Code

ex31: P_2/P_1 Stokes Problem with Temperature on Unstructured Mesh

Additive Schwarz + Full Schur Complement, Elman, Howle, Shadid, Shuttleworth, and Tuminaro, SISC, 2006.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type additive
-pc_fieldsplit_0_fields 0,1 -pc_fieldsplit_1_fields 2
 -fieldsplit_0_ksp_type fgmres -fieldsplit_0_pc_type fieldsplit
 -fieldsplit_0_pc_fieldsplit_type schur
 -fieldsplit_0_pc_fieldsplit_schur_factorization_type full
   -fieldsplit_0_fieldsplit_velocity_ksp_type preonly
   -fieldsplit_0_fieldsplit_velocity_pc_type lu
   -fieldsplit_0_fieldsplit_pressure_ksp_rtol 1e-10
   -fieldsplit_0_fieldsplit_pressure_pc_type jacobi
 -fieldsplit_temperature_ksp_type preonly
 -fieldsplit_temperature_pc_type lu
```

$$\begin{pmatrix} \begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} & 0 \\ 0 & L_T \end{pmatrix}$$

Solver Configuration: No New Code

ex31: P_2/P_1 Stokes Problem with Temperature on Unstructured Mesh

Upper Schur Comp. + Full Schur Comp. + Least-Squares Comm.

```
-ksp_type fgmres -pc_type fieldsplit -pc_fieldsplit_type schur
-pc_fieldsplit_0_fields 0,1 -pc_fieldsplit_1_fields 2
-pc_fieldsplit_schur_factorization_type upper
 -fieldsplit_0_ksp_type fgmres -fieldsplit_0_pc_type fieldsplit
 -fieldsplit_0_pc_fieldsplit_type schur
 -fieldsplit_0_pc_fieldsplit_schur_factorization_type full
   -fieldsplit_0_fieldsplit_velocity_ksp_type preonly
   -fieldsplit_0_fieldsplit_velocity_pc_type lu
   -fieldsplit_0_fieldsplit_pressure_ksp_rtol 1e-10
   -fieldsplit_0_fieldsplit_pressure_pc_type jacobi
 -fieldsplit_temperature_ksp_type gmres
 -fieldsplit_temperature_pc_type lsc
```

$$\begin{pmatrix} \begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} & G \\ 0 & \hat{S}_{\mathrm{LSC}} \end{pmatrix}$$

SNES ex62

Preconditioning

Jacobi

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_gmres_restart 100 -ksp_rtol 1.0e-9
    -ksp_monitor_short
  -pc_type jacobi
```

SNES ex62

Preconditioning

Block diagonal

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type additive
  -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_pc_type jacobi
```

SNES ex62

Preconditioning

Block triangular

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type multiplicative
  -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_pc_type jacobi
```

SNES ex62

Preconditioning

Diagonal Schur complement

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type schur
    -pc_fieldsplit_schur_factorization_type diag
  -fieldsplit_velocity_ksp_type gmres
    -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_ksp_rtol 1e-10
    -fieldsplit_pressure_pc_type jacobi
```

SNES ex62

Preconditioning

Upper triangular Schur complement

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type schur
    -pc_fieldsplit_schur_factorization_type upper
  -fieldsplit_velocity_ksp_type gmres
    -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_ksp_rtol 1e-10
    -fieldsplit_pressure_pc_type jacobi
```

SNES ex62

Preconditioning

Lower triangular Schur complement

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type schur
    -pc_fieldsplit_schur_factorization_type lower
  -fieldsplit_velocity_ksp_type gmres
    -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_ksp_rtol 1e-10
    -fieldsplit_pressure_pc_type jacobi
```

SNES ex62

Preconditioning

Full Schur complement

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -vel_petscspace_order 2 -pres_petscspace_order 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type schur
    -pc_fieldsplit_schur_factorization_type full
  -fieldsplit_velocity_ksp_type gmres
    -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_ksp_rtol 1e-10
    -fieldsplit_pressure_pc_type jacobi
```

Programming with Options

ex55: Allen-Cahn problem in 2D

- constant mobility
- triangular elements

Geometric multigrid method for saddle point variational inequalities:

```
./ex55 -ksp_type fgmres -pc_type mg -mg_levels_ksp_type fgmres
-mg_levels_pc_type fieldsplit -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_pc_fieldsplit_type schur -da_grid_x 65 -da_grid_y 65
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition user
-mg_levels_fieldsplit_1_ksp_type gmres -mg_coarse_ksp_type preonly
-mg_levels_fieldsplit_1_pc_type none   -mg_coarse_pc_type svd
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor    -pc_mg_levels 5
-mg_levels_fieldsplit_0_pc_sor_forward -pc_mg_galerkin
-snes_vi_monitor -ksp_monitor_true_residual -snes_atol 1.e-11
-mg_levels_ksp_monitor -mg_levels_fieldsplit_ksp_monitor
-mg_levels_ksp_max_it 2 -mg_levels_fieldsplit_ksp_max_it 5
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
  -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Schur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Schur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Schur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

Programming with Options

ex55: Allen-Cahn problem in 2D

Smoother: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

Schur complement action: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```

Null spaces

For a single matrix, use

```
MatSetNullSpace(J, nullSpace);
```

to alter the **KSP**, and

```
MatSetNearNullSpace(J, nearNullSpace);
```

to set the coarse basis for AMG.

But this will not work for dynamically created operators.

Null spaces

For a single matrix, use

```
MatSetNullSpace(J, nullSpace);
```

to alter the **KSP**, and

```
MatSetNearNullSpace(J, nearNullSpace);
```

to set the coarse basis for AMG.

But this will not work for dynamically created operators.

Null spaces

Field Split

Can attach a nullspace to the **IS** that creates a split,

```
PetscObjectCompose(pressureIS, "nullspace",
                   (PetscObject) nullSpacePres);
```

If the **DM** makes the **IS**, use

```
PetscObject pressure;

DMGetField(dm, 1, &pressure);
PetscObjectCompose(pressure, "nullspace",
                   (PetscObject) nullSpacePres);
```

Outline

AMG

Why not use AMG?

- Of course we will try AMG
  - GAMG, `-pc_type gamg`
  - ML, `-download-ml`, `-pc_type ml`
  - BoomerAMG, `-download-hypre`, `-pc_type hypre -pc_hypre_type boomeramg`
- Problems with
  - vector character
  - anisotropy
  - scalability of setup time

AMG

Why not use AMG?

- Of course we will try AMG
  - GAMG, `-pc_type gamg`
  - ML, `-download-ml`, `-pc_type ml`
  - BoomerAMG, `-download-hypre`, `-pc_type hypre -pc_hypre_type boomeramg`
- Problems with
  - vector character
  - anisotropy
  - scalability of setup time

AMG

Why not use AMG?

- Of course we will try AMG
  - GAMG, `-pc_type gamg`
  - ML, `-download-ml`, `-pc_type ml`
  - BoomerAMG, `-download-hypre`, `-pc_type hypre -pc_hypre_type boomeramg`
- Problems with
  - vector character
  - anisotropy
  - scalability of setup time

Multigrid with DM

Allows multigrid with some simple command line options

- `-pc_type mg`, `-pc_mg_levels`
- `-pc_mg_type`, `-pc_mg_cycle_type`, `-pc_mg_galerkin`
- `-mg_levels_1_ksp_type`, `-mg_levels_1_pc_type`
- `-mg_coarse_ksp_type`, `-mg_coarse_pc_type`
- `-da_refine`, `-ksp_view`

Interface also works with GAMG and 3rd party packages like ML

A 2D Problem

Problem has:

- 1,640,961 unknowns (on the fine level)
- 8,199,681 nonzeros

	Options	Explanation
`./ex5`	`-da_grid_x 21 -da_grid_y 21`	Original grid is 21x21
	`-ksp_rtol 1.0e-9`	Solver tolerance
	`-da_refine 6`	6 levels of refinement
	`-pc_type mg`	4 levels of multigrid
	`-pc_mg_levels 4`	
	`-snes_monitor -snes_view`	Describe solver

A 3D Problem

Problem has:

- 1,689,600 unknowns (on the fine level)
- 89,395,200 nonzeros

	Options	Explanation
`./ex48`	`-M 5 -N 5`	Coarse problem size
	`-da_refine 5`	5 levels of refinement
	`-ksp_rtol 1.0e-9`	Solver tolerance
	`-thi_mat_type baij`	Needs SOR
	`-pc_type mg`	4 levels of multigrid
	`-pc_mg_levels 4`	
	`-snes_monitor -snes_view`	Describe solver

Outline

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)

Always use **SNES**

Always use **SNES** instead of **KSP**:

- No more costly than linear solver
- Can accomodate unanticipated nonlinearities
- Automatic iterative refinement
- Callback interface can take advantage of problem structure

Jed actually recommends **TS**...

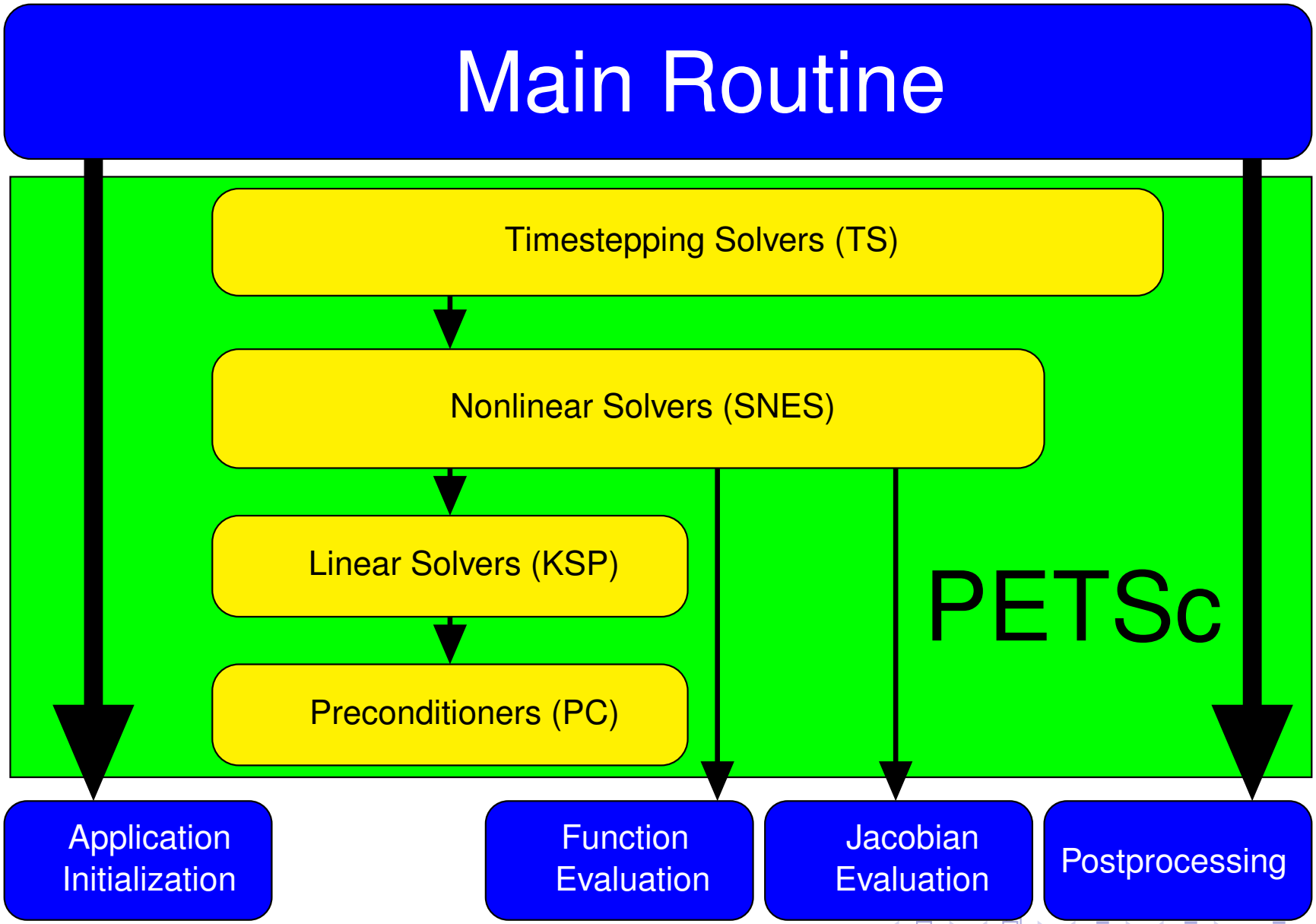
Always use **SNES**

Always use **SNES** instead of **KSP**:

- No more costly than linear solver
- Can accomodate unanticipated nonlinearities
- Automatic iterative refinement
- Callback interface can take advantage of problem structure

Jed actually recommends **TS**...

Flow Control for a PETSc Application



SNES Paradigm

The SNES interface is based upon callback functions

- FormFunction(), set by SNESSetFunction()
- FormJacobian(), set by SNESSetJacobian()

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

- Solver calls the **user's** function
- User function gets application state through the `ctx` variable
  - PETSc never sees application data

SNES Function

User provided function calculates the nonlinear residual:

```
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

User provided function calculates the Jacobian:

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

- `x`: The current solution
- `J`: The Jacobian
- `M`: The Jacobian preconditioning matrix (possibly J itself)
- `ctx`: The user context passed to SNESSetJacobian()
  - Use this to pass application information, e.g. physical constants

Alternatively, you can use

- matrix-free finite difference approximation, `-snes_mf`
- finite difference approximation with coloring, `-snes_fd`

SNES Variants

- Picard iteration
- Line search/Trust region strategies
- Quasi-Newton
- Nonlinear CG/GMRES
- Nonlinear GS/ASM
- Nonlinear Multigrid (FAS)
- Variational inequality approaches

New methods in SNES

LS, TR	Newton-type with line search and trust region
NRichardson	Nonlinear Richardson, usually preconditioned
VIRS, VISS	reduced space and semi-smooth methods for variational inequalities
QN	Quasi-Newton methods like BFGS
NGMRES	Nonlinear GMRES
NCG	Nonlinear Conjugate Gradients
SORQN	SOR quasi-Newton
GS	Nonlinear Gauss-Seidel sweeps
FAS	Full approximation scheme (nonlinear multigrid)
MS	Multi-stage smoothers (in FAS for hyperbolic problems)
Shell	Your method, often used as a (nonlinear) preconditioner

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 (default)
  - 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()

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e2
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view
```

$$
\begin{aligned}
-\Delta U - \partial_y \Omega &= 0 \\
-\Delta V + \partial_x \Omega &= 0 \\
-\Delta \Omega + \nabla \cdot ([U\Omega, V\Omega]) - \mathrm{Gr}\, \partial_x T &= 0 \\
-\Delta T + \mathrm{Pr}\, \nabla \cdot ([UT, VT]) &= 0
\end{aligned}
$$

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e2
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 100
  0 SNES Function norm 768.116
  1 SNES Function norm 658.288
  2 SNES Function norm 529.404
  3 SNES Function norm 377.51
  4 SNES Function norm 304.723
  5 SNES Function norm 2.59998
  6 SNES Function norm 0.00942733
  7 SNES Function norm 5.20667e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e4
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view
```

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e4
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 10000
  0 SNES Function norm 785.404
  1 SNES Function norm 663.055
  2 SNES Function norm 519.583
  3 SNES Function norm 360.87
  4 SNES Function norm 245.893
  5 SNES Function norm 1.8117
  6 SNES Function norm 0.00468828
  7 SNES Function norm 4.417e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view
```

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5
  -da_grid_x 16 -da_grid_y 16 -da_refine 2
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 100000
  0 SNES Function norm 1809.96
Nonlinear solve did not converge due to DIVERGED_LINEAR_SOLVE iterations 0
```

Driven Cavity Problem

SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5
  -da_grid_x 16 -da_grid_y 16 -da_refine 2 -pc_type lu
  -snes_monitor_short -snes_converged_reason -snes_view

lid velocity = 100, prandtl # = 1, grashof # = 100000
  0 SNES Function norm 1809.96
  1 SNES Function norm 1678.37
  2 SNES Function norm 1643.76
  3 SNES Function norm 1559.34
  4 SNES Function norm 1557.6
  5 SNES Function norm 1510.71
  6 SNES Function norm 1500.47
  7 SNES Function norm 1498.93
  8 SNES Function norm 1498.44
  9 SNES Function norm 1498.27
 10 SNES Function norm 1498.18
 11 SNES Function norm 1498.12
 12 SNES Function norm 1498.11
 13 SNES Function norm 1498.11
 14 SNES Function norm 1498.11
 ...
```

Why isn't SNES converging?

- The Jacobian is wrong (maybe only in parallel)
  - Check with `-snes_check_jacobian -snes_check_jacobian_view`
- The linear system is not solved accurately enough
  - Check with `-pc_type lu`
  - Check `-ksp_monitor_true_residual`, try right preconditioning
- The Jacobian is singular with inconsistent right side
  - Use **MatNullSpace** to inform the **KSP** of a known null space
  - Use a different Krylov method or preconditioner
- The nonlinearity is just really strong
  - Run with `-info` or `-snes_ls_monitor` to see line search
  - Try using trust region instead of line search `-snes_type tr`
  - Try grid sequencing if possible `-snes_grid_sequence`
  - Use a continuation

Nonlinear Preconditioning

PC preconditions **KSP**

```
-ksp_type gmres
-pc_type richardson
```

SNES preconditions **SNES**

```
-snes_type ngmres
-npc_snes_type nrichardson
```

Nonlinear Preconditioning

PC preconditions **KSP**

```
-ksp_type gmres

-pc_type richardson
```

SNES preconditions **SNES**

```
-snes_type ngmres

-npc_snes_type nrichardson
```

Nonlinear Use Cases

Warm start Newton

```
-snes_type newtonls
-npc_snes_type nrichardson -npc_snes_max_it 5
```

Cleanup noisy Jacobian

```
-snes_type ngmres -snes_ngmres_m 5
-npc_snes_type newtonls
```

Additive-Schwarz Preconditioned Inexact Newton

```
-snes_type aspin -snes_npc_side left
-npc_snes_type nasm -npc_snes_nasm_type restrict
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type newtonls -snes_converged_reason
 -pc_type lu
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 1132.29
  2 SNES Function norm 1026.17
  3 SNES Function norm 925.717
  4 SNES Function norm 924.778
  5 SNES Function norm 836.867
  :
 21 SNES Function norm 585.143
 22 SNES Function norm 585.142
 23 SNES Function norm 585.142
 24 SNES Function norm 585.142
  :
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type fas -snes_converged_reason
 -fas_levels_snes_type gs -fas_levels_snes_max_it 6
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 574.793
  2 SNES Function norm 513.02
  3 SNES Function norm 216.721
  4 SNES Function norm 85.949
Nonlinear solve did not converge due to DIVERGED_INNER iterations 4
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type fas -snes_converged_reason
 -fas_levels_snes_type gs -fas_levels_snes_max_it 6
  -fas_coarse_snes_converged_reason
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 12
  1 SNES Function norm 574.793
    Nonlinear solve did not converge due to DIVERGED_MAX_IT its 50
  2 SNES Function norm 513.02
    Nonlinear solve did not converge due to DIVERGED_MAX_IT its 50
  3 SNES Function norm 216.721
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 22
  4 SNES Function norm 85.949
    Nonlinear solve did not converge due to DIVERGED_LINE_SEARCH its 42
Nonlinear solve did not converge due to DIVERGED_INNER iterations 4
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type fas -snes_converged_reason
 -fas_levels_snes_type gs -fas_levels_snes_max_it 6
  -fas_coarse_snes_linesearch_type basic
  -fas_coarse_snes_converged_reason


lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  ⋮
 47 SNES Function norm 78.8401
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 5
 48 SNES Function norm 73.1185
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
 49 SNES Function norm 78.834
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 5
 50 SNES Function norm 73.1176
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  ⋮
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type nrichardson -npc_snes_max_it 1 -snes_converged_reason
 -npc_snes_type fas -npc_fas_coarse_snes_converged_reason
  -npc_fas_levels_snes_type gs -npc_fas_levels_snes_max_it 6
  -npc_fas_coarse_snes_linesearch_type basic
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  1 SNES Function norm 552.271
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 27
  2 SNES Function norm 173.45
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 45
  :
 43 SNES Function norm 3.45407e-05
    Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
 44 SNES Function norm 1.6141e-05
    Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
 45 SNES Function norm 9.13386e-06
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 45
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type ngmres -npc_snes_max_it 1 -snes_converged_reason
 -npc_snes_type fas -npc_fas_coarse_snes_converged_reason
  -npc_fas_levels_snes_type gs -npc_fas_levels_snes_max_it 6
  -npc_fas_coarse_snes_linesearch_type basic
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  1 SNES Function norm 538.605
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 13
  2 SNES Function norm 178.005
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 24
  :
 27 SNES Function norm 0.000102487
    Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 2
 28 SNES Function norm 4.2744e-05
    Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
 29 SNES Function norm 1.01621e-05
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 29
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type ngmres -npc_snes_max_it 1 -snes_converged_reason
 -npc_snes_type fas -npc_fas_coarse_snes_converged_reason
 -npc_fas_levels_snes_type newtonls -npc_fas_levels_snes_max_it 6
  -npc_fas_levels_snes_linesearch_type basic
  -npc_fas_levels_snes_max_linear_solve_fail 30
  -npc_fas_levels_ksp_max_it 20 -npc_fas_levels_snes_converged_reason
  -npc_fas_coarse_snes_linesearch_type basic
lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
    Nonlinear solve did not converge due to DIVERGED_MAX_IT its 6
    :
    :
        Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 1
    :
    :
  1 SNES Function norm 0.1935
  2 SNES Function norm 0.0179938
  3 SNES Function norm 0.00223698
  4 SNES Function norm 0.000190461
  5 SNES Function norm 1.6946e-06
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 5
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type composite -snes_composite_type additiveoptimal
 -snes_composite_sneses fas,newtonls -snes_converged_reason
 -sub_0_fas_levels_snes_type gs -sub_0_fas_levels_snes_max_it 6
   -sub_0_fas_coarse_snes_linesearch_type basic
 -sub_1_snes_linesearch_type basic -sub_1_pc_type mg


lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 541.462
  2 SNES Function norm 162.92
  3 SNES Function norm 48.8138
  4 SNES Function norm 11.1822
  5 SNES Function norm 0.181469
  6 SNES Function norm 0.00170909
  7 SNES Function norm 3.24991e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

Nonlinear Preconditioning

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
 -snes_type composite -snes_composite_type multiplicative
 -snes_composite_sneses fas,newtonls -snes_converged_reason
 -sub_0_fas_levels_snes_type gs -sub_0_fas_levels_snes_max_it 6
   -sub_0_fas_coarse_snes_linesearch_type basic
 -sub_1_snes_linesearch_type basic -sub_1_pc_type mg


lid velocity = 100, prandtl # = 1, grashof # = 50000
  0 SNES Function norm 1228.95
  1 SNES Function norm 544.404
  2 SNES Function norm 18.2513
  3 SNES Function norm 0.488689
  4 SNES Function norm 0.000108712
  5 SNES Function norm 5.68497e-08
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 5
```

Nonlinear Preconditioning

Solver	T	N. It	L. It	Func	Jac	PC	NPC
$(\mathcal{N}\backslash \mathrm{K}-\mathrm{MG})$	9.83	17	352	34	85	370	–
$\mathrm{NGMRES}-_R$ $(\mathcal{N}\backslash \mathrm{K}-\mathrm{MG})$	7.48	10	220	21	50	231	10
FAS	6.23	162	0	2382	377	754	–
$\mathrm{FAS}+(\mathcal{N}\backslash \mathrm{K}-\mathrm{MG})$	8.07	10	197	232	90	288	–
$\mathrm{FAS}*(\mathcal{N}\backslash \mathrm{K}-\mathrm{MG})$	4.01	5	80	103	45	125	–
$\mathrm{NRICH}-_L \mathrm{FAS}$	3.20	50	0	1180	192	384	50
$\mathrm{NGMRES}-_R \mathrm{FAS}$	1.91	24	0	447	83	166	24

Nonlinear Preconditioning

See discussion in:

Composing Scalable Nonlinear Algebraic Solvers,
Peter Brune, Matthew Knepley, Barry Smith, and Xuemin Tu,

SIAM Review, **57**(4), 535–565, 2015.

http://www.mcs.anl.gov/uploads/cels/papers/P2010-0112.pdf

Hierarchical Krylov

This tests a hierarchical Krylov method

```
mpiexec -n 4 ./ex19 -da_refine 4 -snes_view
  -ksp_type fgmres -pc_type bjacobi -pc_bjacobi_blocks 2
  -sub_ksp_type gmres -sub_pc_type bjacobi -sub_ksp_max_it 2
  -sub_sub_ksp_type preonly -sub_sub_pc_type ilu
```

```
SNES Object: 4 MPI processes
  type: newtonls
  KSP Object:   4 MPI processes
    type: fgmres
  PC Object:   4 MPI processes
    type: bjacobi
      block Jacobi: number of blocks = 2
    KSP Object:(sub_) 2 MPI processes
      type: gmres
    PC Object:(sub_) 2 MPI processes
      type: bjacobi
        block Jacobi: number of blocks = 2
      KSP Object:  (sub_sub_)   1 MPI processes
        type: preonly
      PC Object:  (sub_sub_)   1 MPI processes
        type: ilu
          ILU: out-of-place factorization
```

Hierarchical Krylov

This tests a hierarchical Krylov method

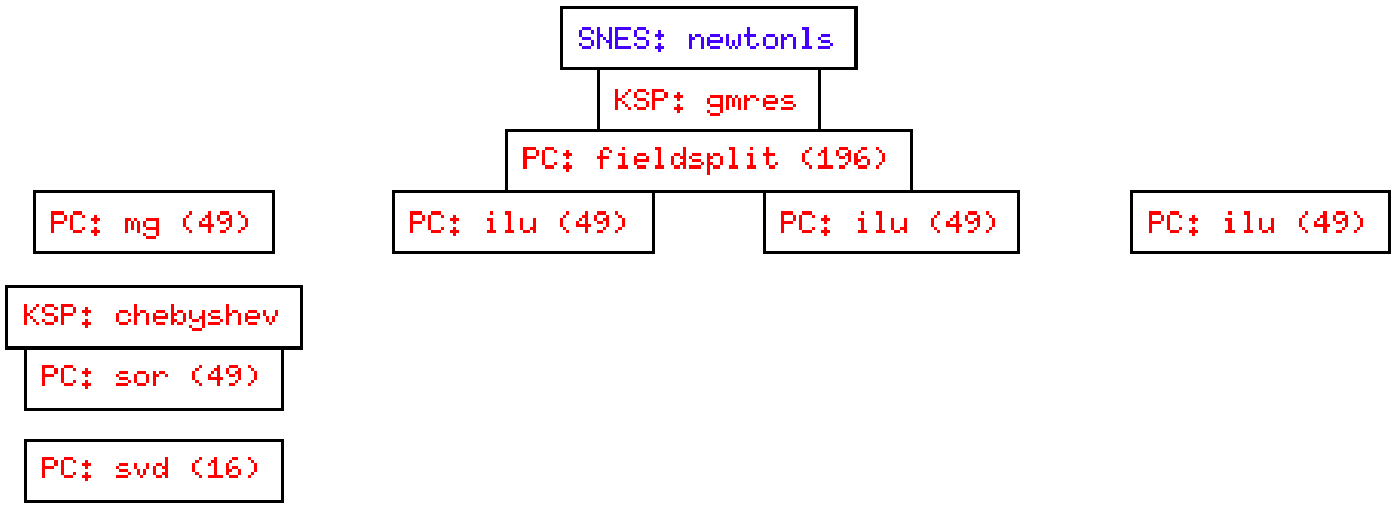
```
mpiexec -n 4 ./ex19 -da_refine 4 -snes_view
  -ksp_type fgmres -pc_type bjacobi -pc_bjacobi_blocks 2
  -sub_ksp_type gmres -sub_pc_type bjacobi -sub_ksp_max_it 2
  -sub_sub_ksp_type preonly -sub_sub_pc_type ilu

  PC Object:   4 MPI processes
    type: bjacobi
      block Jacobi: number of blocks = 2
    PC Object:(sub_) 2 MPI processes
      type: bjacobi
        block Jacobi: number of blocks = 2
      PC Object:  (sub_sub_)   1 MPI processes
        type: ilu
          ILU: out-of-place factorization
        Mat Object:     1 MPI processes
          type: seqaij
          rows=2500, cols=2500, bs=4
      Mat Object:   2 MPI processes
        type: mpiaij
        rows=4900, cols=4900, bs=4
    Mat Object:     4 MPI processes
      type: mpiaij
      rows=9604, cols=9604, bs=4
```

Visualizing Solvers

This shows how to visualize a nested solver configuration:

```
./ex19 -da_refine 1 -pc_type fieldsplit -fieldsplit_x_velocity_pc_type mg
  -fieldsplit_x_velocity_mg_coarse_pc_type svd
  -snes_view draw -draw_pause -2 -geometry 0,0,600,600
```



Outline

What about TS?

Time Integration in PETSc Has Made a Quantum Leap

and is now a premier library in the field,

thanks to Jed, Emil, and Peter $\Longrightarrow$

What about TS?

Time Integration in PETSc Has Made a Quantum Leap

and is now a premier library in the field,

thanks to Jed, Emil, and Peter $\Longrightarrow$

What about TS?

Time Integration in PETSc Has Made a Quantum Leap

and is now a premier library in the field,

thanks to Jed, Emil, and Peter $\Longrightarrow$

Some TS methods

TSSSPRK104 10-stage, fourth order, low-storage, optimal explicit SSP Runge-Kutta $c_{\text{eff}} = 0.6$ (Ketcheson 2008)

TSARKIMEX2E second order, one explicit and two implicit stages, L-stable, optimal (Constantinescu)

TSARKIMEX3 (and 4 and 5), L-stable (Kennedy and Carpenter, 2003)

TSROSWRA3PW three stage, third order, for index-1 PDAE, A-stable, $R(\infty) = 0.73$, second order strongly A-stable embedded method (Rang and Angermann, 2005)

TSROSWRA34PW2 four stage, third order, L-stable, for index 1 PDAE, second order strongly A-stable embedded method (Rang and Angermann, 2005)

TSROSWLLSSP3P4S2C four stage, third order, L-stable implicit, SSP explicit, L-stable embedded method (Constantinescu)

IMEX time integration in PETSc

Additive Runge-Kutta IMEX methods

$$G(t, x, \dot{x}) = F(t, x)$$
$$J_\alpha = \alpha G_{\dot{x}} + G_x$$

User provides:

- `FormRHSFunction(ts,t,x,F,void *ctx)`
- `FormIFunction(ts,t,x,xdot,G,void *ctx)`
- `FormIJacobian(ts,t,x,xdot,alpha,J,J_p,mstr,void *ctx)`

- Single step interface so user can have own time loop
- Choice of explicit method, e.g. SSP
- L-stable DIRK for stiff part G
- Orders 2 through 5, embedded error estimates
- Dense output, hot starts for Newton
- More accurate methods if G is linear, also Rosenbrock-W
- Can use preconditioner from classical “semi-implicit” methods
- Extensible adaptive controllers, can change order within a family
- Easy to register new methods: `TSARKIMEXRegister()`

Stiff linear advection-reaction test problem

Equations

TS ex22.c

$$u_t + a_1 u_x = -k_1 u + k_2 v + s_1$$
$$v_t + a_2 v_x = \quad k_1 u - k_2 v + s_2$$

Upstream boundary condition:

$$u(0, t) = 1 - \sin(12t)^4$$

Stiff linear advection-reaction test problem

Equations

TS ex22.c

$$u_t + a_1 u_x = -k_1 u + k_2 v + s_1$$
$$v_t + a_2 v_x = \quad k_1 u - k_2 v + s_2$$

```
FormIFunction(TS ts, PetscReal t, Vec X, Vec Xdot, Vec F, void *ptr) {
  TSGetDM(ts, &da);
  DMDAGetLocalInfo(da, &info);
  DMDAVecGetArray(da, X, &x);
  DMDAVecGetArray(da, Xdot, &xdot);
  DMDAVecGetArray(da, F, &f);
  /* Compute function over the locally owned part of the grid */
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    f[i][0] = xdot[i][0] + k[0]*x[i][0] - k[1]*x[i][1] - s[0];
    f[i][1] = xdot[i][1] - k[0]*x[i][0] + k[1]*x[i][1] - s[1];
  }
  DMDAVecRestoreArray(da, X, &x);
  DMDAVecRestoreArray(da, Xdot, &xdot);
  DMDAVecRestoreArray(da, F, &f);
}
```

Stiff linear advection-reaction test problem

Equations

TS ex22.c

$$u_t + a_1 u_x = -k_1 u + k_2 v + s_1$$
$$v_t + a_2 v_x = \quad k_1 u - k_2 v + s_2$$

```
FormIJacobian(TS ts, PetscReal t, Vec X, Vec Xdot, PetscReal a, Mat *J,
              Mat *Jpre, MatStructure *str, void *ptr) {
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    PetscScalar v[2][2];
    v[0][0] = a + k[0]; v[0][1] =  -k[1];
    v[1][0] =    -k[0]; v[1][1] = a+k[1];
    MatSetValuesBlocked(*Jpre, 1, &i, 1, &i, &v[0][0], INSERT_VALUES);
  }
  MatAssemblyBegin(*Jpre, MAT_FINAL_ASSEMBLY);
  MatAssemblyEnd(*Jpre, MAT_FINAL_ASSEMBLY);
  if (*J != *Jpre) {
    MatAssemblyBegin(*J, MAT_FINAL_ASSEMBLY);
    MatAssemblyEnd(*J, MAT_FINAL_ASSEMBLY);
  }
}
```

Stiff linear advection-reaction test problem

Equations

TS ex22.c

$$u_t + a_1 u_x = -k_1 u + k_2 v + s_1$$
$$v_t + a_2 v_x = \quad k_1 u - k_2 v + s_2$$

```
FormRHSFunction(TS ts, PetscReal t, Vec X, Vec F, void *ptr) {
  PetscReal u0t[2] = {1. - PetscPowScalar(sin(12*t),4.),0};
  DMGetLocalVector(da, &Xloc);
  DMGlobalToLocalBegin(da, X, INSERT_VALUES, Xloc);
  DMGlobalToLocalEnd(da, X, INSERT_VALUES, Xloc);
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    /* CALCULATE RESIDUAL f[i][j] */
  }
}
```

Stiff linear advection-reaction test problem

Equations

TS ex22.c

$$u_t + a_1 u_x = -k_1 u + k_2 v + s_1$$
$$v_t + a_2 v_x = \quad k_1 u - k_2 v + s_2$$

```
for (i = info.xs; i < info.xs+info.xm; ++i) {
for (j = 0; j < 2; ++j) {
  const PetscReal a = a[j]/hx;
  if (i == 0)                  f[i][j] =
    a*(1/3*u0t[j] + 1/2*x[i][j] - x[i+1][j] + 1/6*x[i+2][j]);
  else if (i == 1)             f[i][j] =
    a*(-1/12*u0t[j] + 2/3*x[i-1][j] - 2/3*x[i+1][j] + 1/12*x[i+2][j]);
  else if (i == info.mx-2) f[i][j] =
    a*(-1/6*x[i-2][j] + x[i-1][j] - 1/2*x[i][j] - 1/3*x[i+1][j]);
  else if (i == info.mx-1) f[i][j] =
    a*(-x[i][j] + x[i-1][j]);
  else                         f[i][j] =
    a*(-1/12*x[i-2][j] + 2/3*x[i-1][j] - 2/3*x[i+1][j] + 1/12*x[i+2][j]);
}
}
```

Stiff linear advection-reaction test problem

Parameters

TS ex22.c

$$a_1 = 1, \quad k_1 = 10^6, \quad s_1 = 0,$$
$$a_2 = 0, \quad k_2 = 2k_1, \quad s_2 = 1$$

Stiff linear advection-reaction test problem

Initial conditions

TS ex22.c

$$u(x,0) = 1 + s_2 x$$

$$v(x,0) = \frac{k_0}{k_1} u(x,0) + \frac{s_1}{k_1}$$

```
PetscErrorCode FormInitialSolution(TS ts, Vec X, void *ctx) {
  TSGetDM(ts, &da);
  DMDAGetLocalInfo(da, &info);
  DMDAVecGetArray(da, X, &x);
  /* Compute function over the locally owned part of the grid */
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    PetscReal r  = (i+1)*hx;
    PetscReal ik = user->k[1] != 0.0 ? 1.0/user->k[1] : 1.0;
    x[i][0] = 1 + user->s[1]*r;
    x[i][1] = user->k[0]*ik*x[i][0] + user->s[1]*ik;
  }
  DMDAVecRestoreArray(da, X, &x);
}
```

Stiff linear advection-reaction test problem

Initial conditions

TS ex22.c

$$u(x,0) = 1 + s_2 x$$

$$v(x,0) = \frac{k_0}{k_1} u(x,0) + \frac{s_1}{k_1}$$

```
PetscErrorCode FormInitialSolution(TS ts, Vec X, void *ctx) {
  TSGetDM(ts, &da);
  DMDAGetLocalInfo(da, &info);
  DMDAVecGetArray(da, X, &x);
  /* Compute function over the locally owned part of the grid */
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    PetscReal r  = (i+1)*hx;
    PetscReal ik = user->k[1] != 0.0 ? 1.0/user->k[1] : 1.0;
    x[i][0] = 1 + user->s[1]*r;
    x[i][1] = user->k[0]*ik*x[i][0] + user->s[1]*ik;
  }
  DMDAVecRestoreArray(da, X, &x);
}
```

Stiff linear advection-reaction test problem

Examples

TS ex22.c

- `./ex22 -da_grid_x 200 -ts_monitor_draw_solution -ts_arkimex_type 4 -ts_adapt_type none`
- `./ex22 -da_grid_x 200 -ts_monitor_draw_solution -ts_type rosw -ts_dt 1e-3 -ts_adapt_type none`
- `./ex22 -da_grid_x 200 -ts_monitor_draw_solution -ts_type rosw -ts_rosw_type sandu3 -ts_dt 5e-3 -ts_adapt_type none`
- `./ex22 -da_grid_x 200 -ts_monitor_draw_solution -ts_type rosw -ts_rosw_type ra34pw2 -ts_adapt_monitor`

1D Brusselator reaction-diffusion

Equations

TS ex25.c

$$u_t - \alpha u_{xx} = A - (B+1)u + u^2 v$$
$$v_t - \alpha v_{xx} = Bu - u^2 v$$

Boundary conditions:

$$u(0,t) = u(1,t) = 1$$
$$v(0,t) = v(1,t) = 3$$

1D Brusselator reaction-diffusion

Equations

TS ex25.c

$$u_t - \alpha u_{xx} = A - (B+1)u + u^2 v$$
$$v_t - \alpha v_{xx} = Bu - u^2 v$$

```
FormIFunction(TS ts, PetscReal t, Vec X, Vec Xdot, Vec F, void *ptr) {
  DMGlobalToLocalBegin(da, X, INSERT_VALUES, Xloc);
  DMGlobalToLocalEnd(da, X, INSERT_VALUES, Xloc);
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    if (i == 0) {
      f[i].u = hx * (x[i].u - uleft);
      f[i].v = hx * (x[i].v - vleft);
    } else if (i == info.mx-1) {
      f[i].u = hx * (x[i].u - uright);
      f[i].v = hx * (x[i].v - vright);
    } else {
      f[i].u = hx * xdot[i].u - alpha * (x[i-1].u - 2.*x[i].u + x[i+1].u)
      f[i].v = hx * xdot[i].v - alpha * (x[i-1].v - 2.*x[i].v + x[i+1].v)
    }
  }
```

1D Brusselator reaction-diffusion

Equations

TS ex25.c

$$u_t - \alpha u_{xx} = A - (B+1)u + u^2 v$$
$$v_t - \alpha v_{xx} = Bu - u^2 v$$

```
FormIJacobian(TS ts, PetscReal t, Vec X, Vec Xdot, PetscReal a, Mat *J,
              Mat *Jpre, MatStructure *str, void *ptr) {
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    if (i == 0 || i == info.mx-1) {
      const PetscInt    row = i,col = i;
      const PetscScalar vals[2][2] = {{hx,0},{0,hx}};
      MatSetValuesBlocked(*Jpre,1,&row,1,&col,&vals[0][0],INSERT_VALUES);
    } else {
      const PetscInt    row  = i,col[] = {i-1,i,i+1};
      const PetscScalar dL = -alpha/hx,dC = 2*alpha/hx,dR = -alpha/hx;
      const PetscScalar v[2][3][2] = {{{dL,0},{a*hx+dC,0},{dR,0}},
                                      {{0,dL},{0,a*hx+dC},{0,dR}}};
      MatSetValuesBlocked(*Jpre,1,&row,3,col,&v[0][0][0],INSERT_VALUES);
    }
  }
```

1D Brusselator reaction-diffusion

Equations

TS ex25.c

$$u_t - \alpha u_{xx} = A - (B+1)u + u^2 v$$
$$v_t - \alpha v_{xx} = Bu - u_2^2 v$$

```
FormRHSFunction(TS ts, PetscReal t, Vec X, Vec F, void *ptr) {
  TSGetDM(ts, &da);
  DMDAGetLocalInfo(da, &info);
  DMDAVecGetArray(da, X, &x);
  DMDAVecGetArray(da, F, &f);
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    PetscScalar u = x[i].u, v = x[i].v;
    f[i].u = hx*(A - (B+1)*u + u*u*v);
    f[i].v = hx*(B*u - u*u*v);
  }
  DMDAVecRestoreArray(da, X, &x);
  DMDAVecRestoreArray(da, F, &f);
}
```

1D Brusselator reaction-diffusion

Parameters

TS ex25.c

$$A = 1, \qquad B = 3, \qquad \alpha = 1/50$$

1D Brusselator reaction-diffusion

Initial conditions

TS ex25.c

$$u(x,0) = 1 + \sin(2\pi x)$$
$$v(x,0) = 3$$

```
PetscErrorCode FormInitialSolution(TS ts, Vec X, void *ctx) {
  TSGetDM(ts, &da);
  DMDAGetLocalInfo(da, &info);
  DMDAVecGetArray(da, X, &x);
  /* Compute function over the locally owned part of the grid */
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    PetscReal xi = i*hx;
    x[i].u = uleft*(1-xi) + uright*xi + sin(2*PETSC_PI*xi);
    x[i].v = vleft*(1-xi) + vright*xi;
  }
  DMDAVecRestoreArray(da, X, &x);
}
```

1D Brusselator reaction-diffusion

Initial conditions

TS ex25.c

$$u(x,0) = 1 + \sin(2\pi x)$$
$$v(x,0) = 3$$

```
PetscErrorCode FormInitialSolution(TS ts, Vec X, void *ctx) {
  TSGetDM(ts, &da);
  DMDAGetLocalInfo(da, &info);
  DMDAVecGetArray(da, X, &x);
  /* Compute function over the locally owned part of the grid */
  for (i = info.xs; i < info.xs+info.xm; ++i) {
    PetscReal xi = i*hx;
    x[i].u = uleft*(1-xi) + uright*xi + sin(2*PETSC_PI*xi);
    x[i].v = vleft*(1-xi) + vright*xi;
  }
  DMDAVecRestoreArray(da, X, &x);
}
```

1D Brusselator reaction-diffusion

Examples

TS ex25.c

- `./ex25 -da_grid_x 20 -ts_monitor_draw_solution -ts_type rosw -ts_dt 5e-2 -ts_adapt_type none`
- `./ex25 -da_grid_x 20 -ts_monitor_draw_solution -ts_type rosw -ts_rosw_type 2p -ts_dt 5e-2`
- `./ex25 -da_grid_x 20 -ts_monitor_draw_solution -ts_type rosw -ts_rosw_type 2p -ts_dt 5e-2 -ts_adapt_type none`

Second Order TVD Finite Volume Method

Example

TS ex11.c

- `./ex11 -f $PETSC_DIR/share/petsc/datafiles/meshes/sevenside.exo`
- `./ex11 -f $PETSC_DIR/share/petsc/datafiles/meshes/sevenside-quad-15.exo`
- `./ex11 -f $PETSC_DIR/share/petsc/datafiles/meshes/sevenside.exo -ts_type rosw`

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