The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Computation Institute
University of Chicago

PETSc Tutorial
Groupe Calcul, CNRS University Paris-Sud 11
Orsay, France June 11–13, 2013

Main Point

Never believe *anything*,

unless you can run it.

Main Point

Never believe *anything*,

unless you can run it.

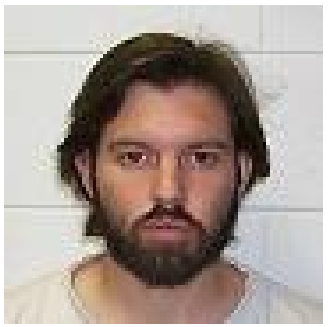
The PETSc Team


Bill Gropp


Barry Smith


Satish Balay


Jed Brown


Matt Knepley

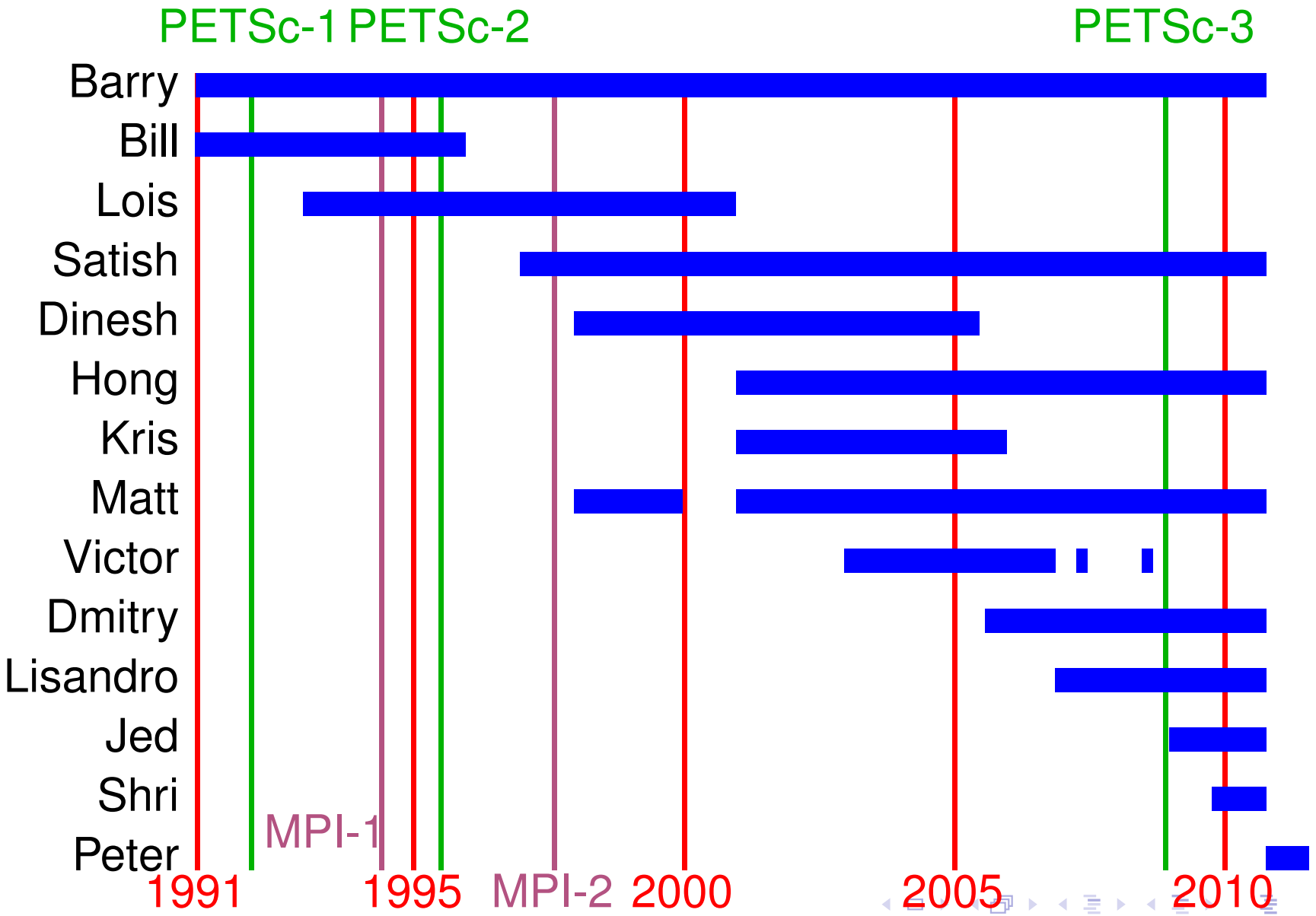

Lisandro Dalcin


Hong Zhang


Mark Adams


Peter Brune

Timeline


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

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

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

Outline

DM Interface

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

DM Interface

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

DM Interface

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

Multigrid Paradigm

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

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

Then **PCMG** organizes

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

Outline

What is a DMDA?

DMDA is a topology interface on structured grids

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

Residual Evaluation

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

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

Callbacks are registered using

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

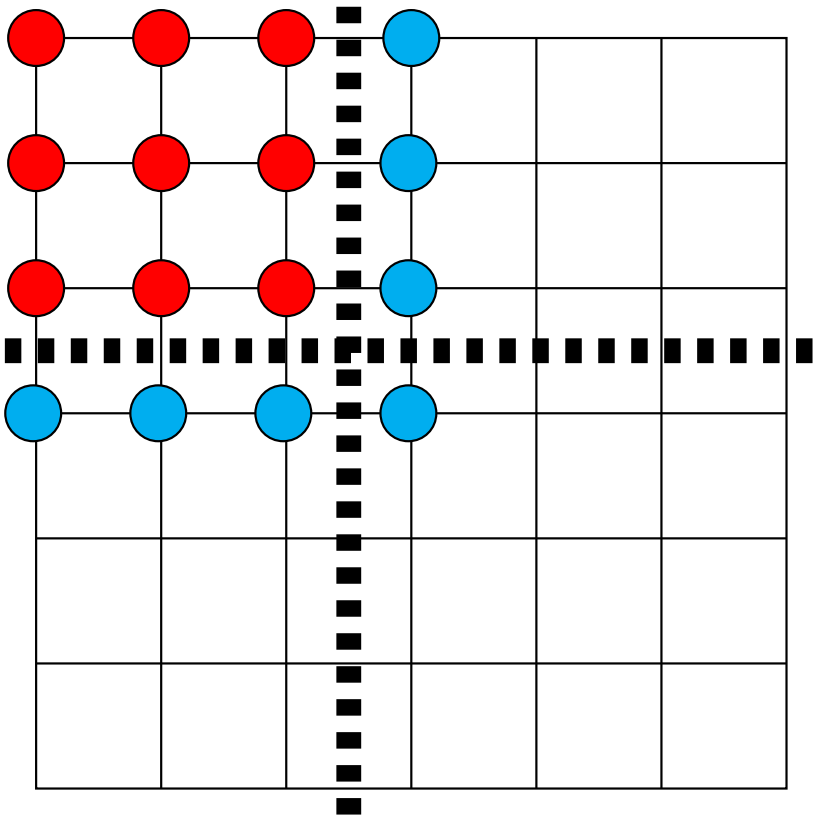
When PETSc needs to evaluate the nonlinear residual **F(x)**,

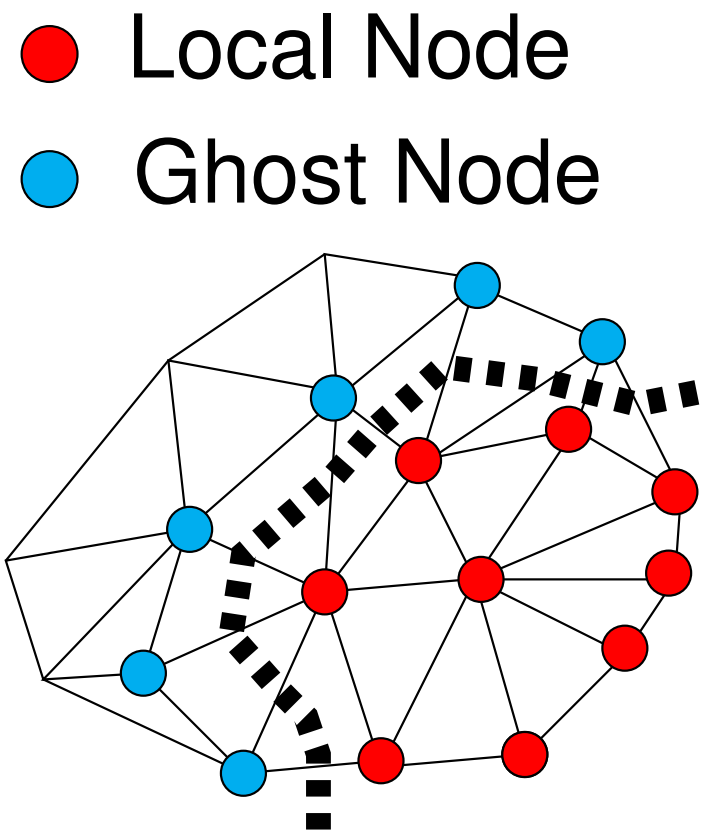
- Each process evaluates the local residual
- PETSc assembles the global residual automatically
  - Uses `DMLocalToGlobal()` method

Ghost Values

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

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





DMDA Global Numberings

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

Natural numbering

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

PETSc numbering

DMDA Global vs. Local Numbering

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

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

Local numbering

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

Global numbering

DMDA Local Function

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

```
(*lfunc)(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 `DASetLocalFunction()`

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] = 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);
}}}
```

$PETCS_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);
}}}}
```

$PETCS_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

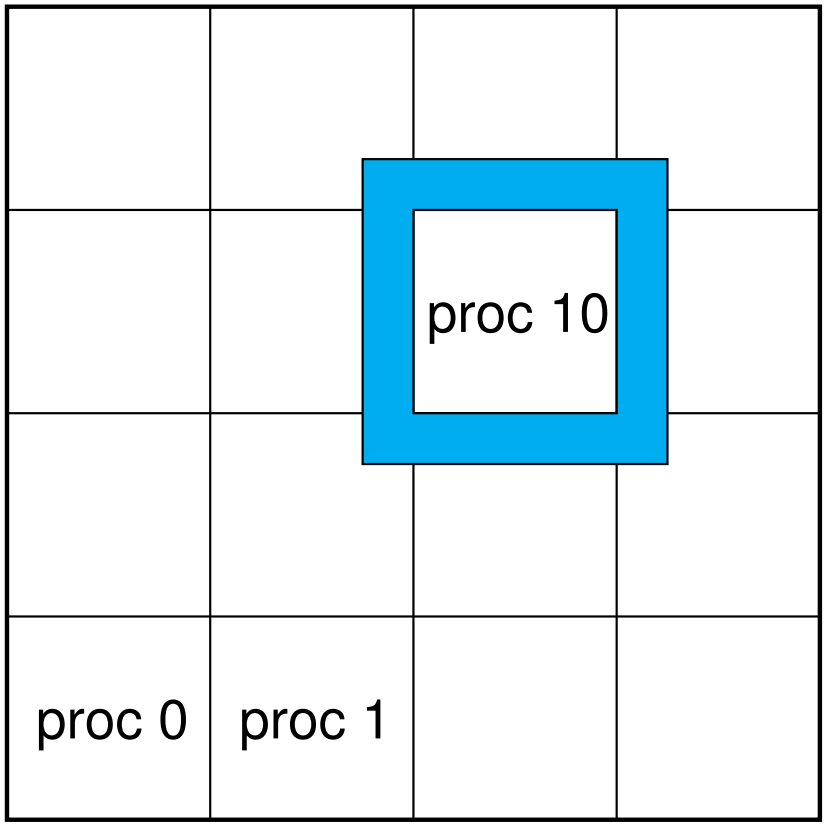
Two-step process enables overlapping computation and communication

- `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

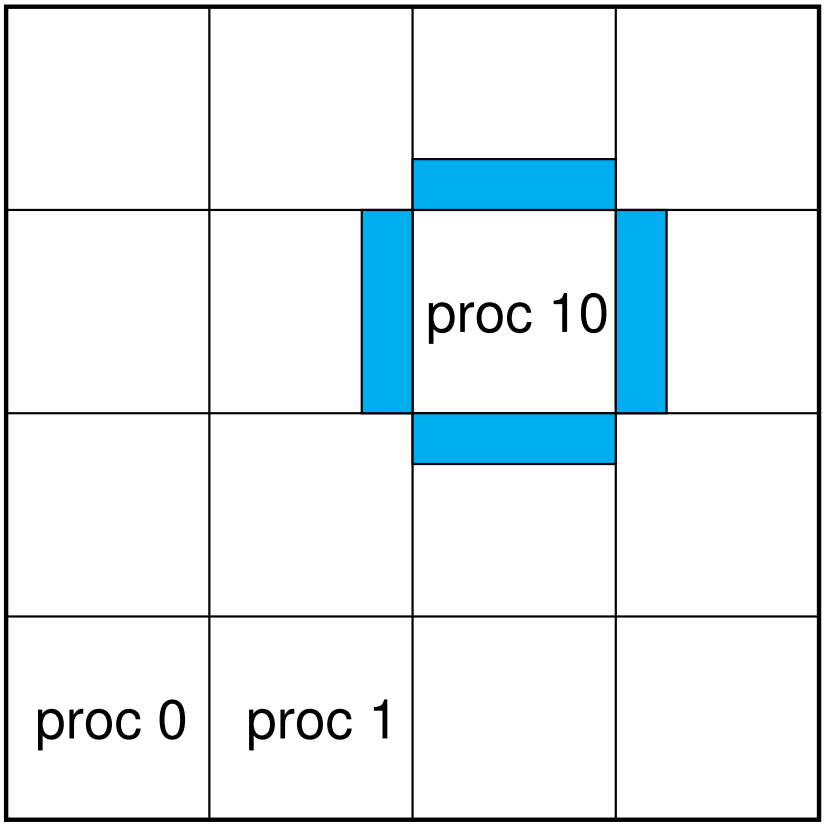
The process can be reversed with `DALocalToGlobalBegin/End()`.

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 *d
```

`bd`: Specifies boundary behavior

- `DMDA_BOUNDARY_NONE`, `DMDA_BOUNDARY_GHOSTED`, or `DMDA_BOUNDARY_PERIODIC`

`type`: Specifies stencil

- `DA_STENCIL_BOX` or `DA_STENCIL_STAR`

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

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

`dof`: Degrees of freedom per node

`s`: The stencil width

`lm/n`: Alternative array of local sizes

- Use **PETSC_NULL** for the default

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

Problem

Traditional PDE codes cannot:

- Compare different discretizations
  - Different orders, finite elements
  - finite volume vs. finite element
- Compare different mesh types
  - Simplicial, hexahedral, polyhedral
- Run 1D, 2D, and 3D problems
- Enable an optimal solver
  - Fields, auxiliary operators

Why?
Impedence Mismatch in Interface

Interface is Too General:

- Solver not told about discretization data, e.g. fields
- Cannot take advantage of problem structure
  - blocking
  - saddle point structure

Interface is Too Specific:

- Assembly code specialized to each discretization
  - dimension, cell shape, hybrid
- Explicit references to element type
  - getVertices(faceID), getAdjacency(edgeID, VERTEX), getAdjacency(edgeID, dim = 0)
- No interface for transitive closure
  - Awkward nested loops to handle different dimensions

Mesh Representation

We represent each mesh as a Hasse Diagram:

- Can represent any CW complex
- Can be implemented as a Directed Acyclic Graph
- Reduces mesh information to a single *covering* relation
- Can discover dimension, since meshes are ranked posets

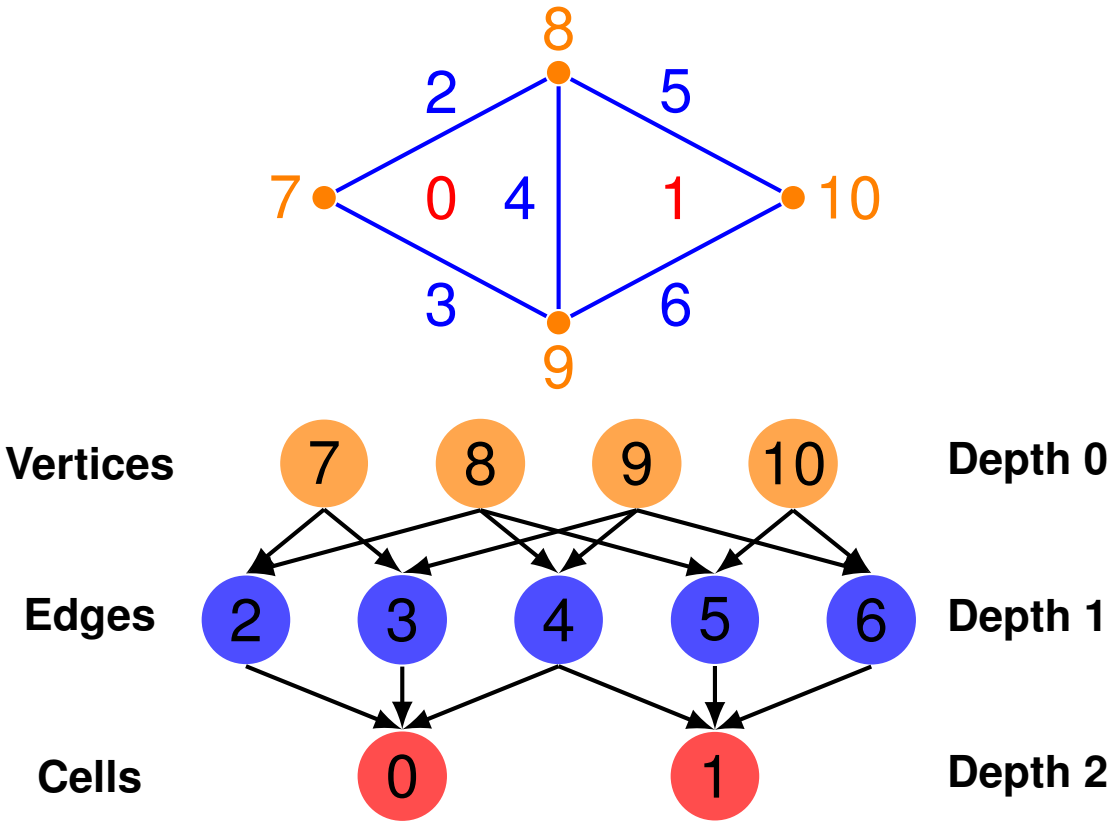
We use an abstract topological interface to organize traversals for:

- discretization integrals
- solver size determination
- computing communication patterns

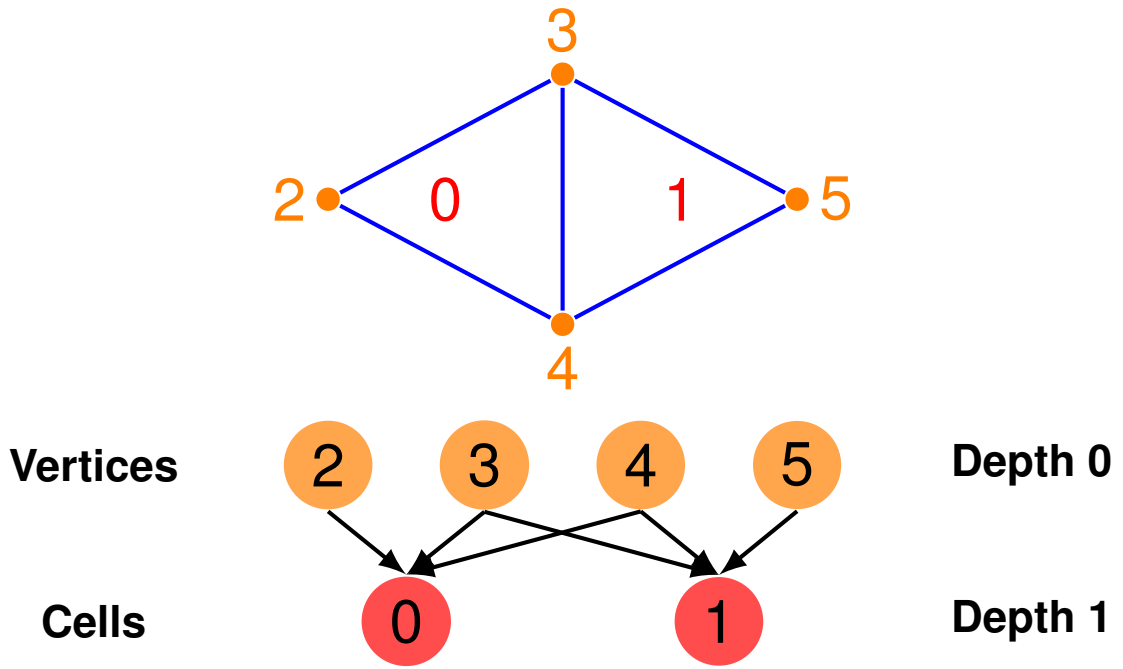
Mesh geometry is treated as just another mesh function.

Sample Meshes

Interpolated triangular mesh

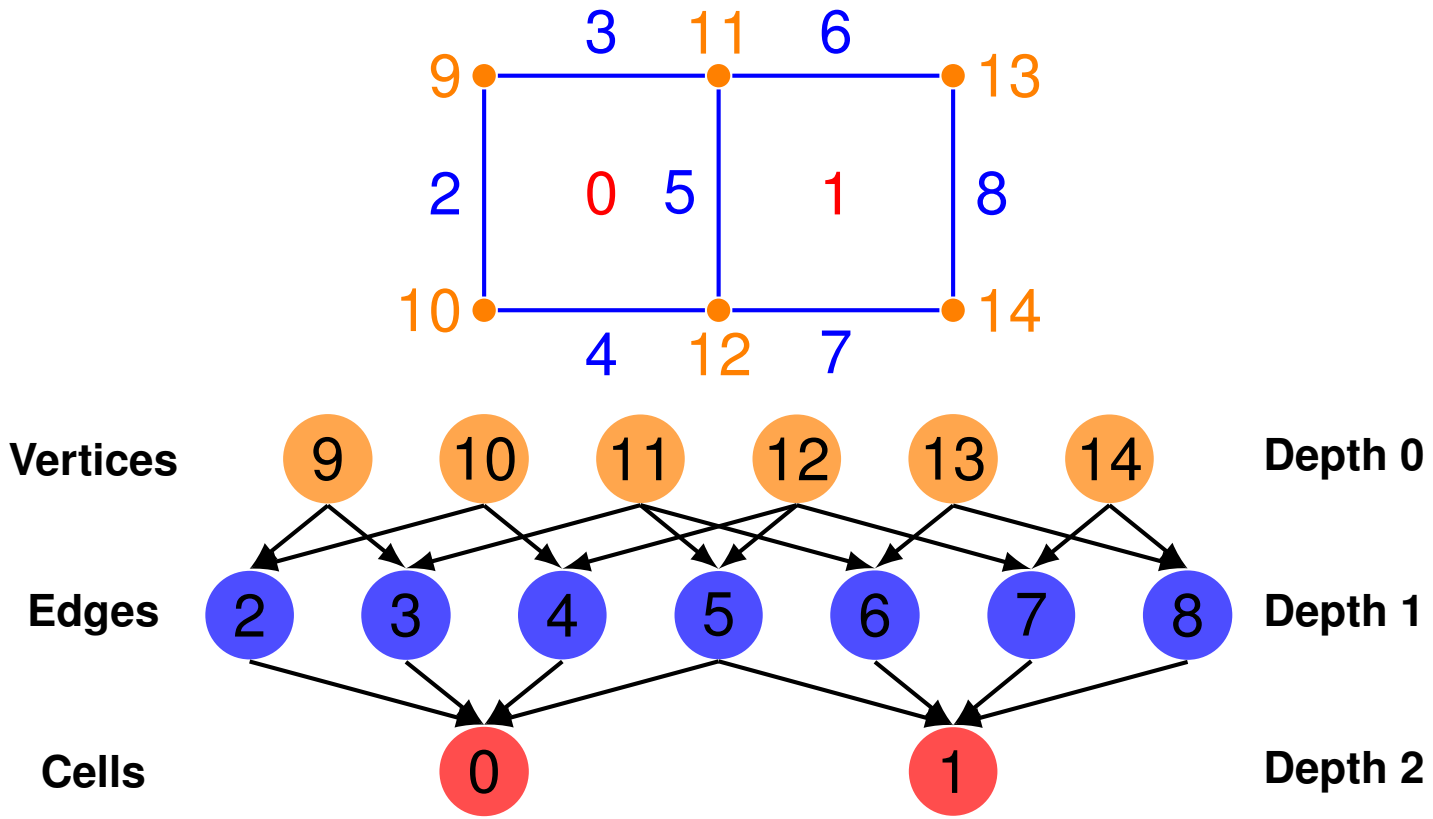


Sample Meshes
Optimized triangular mesh



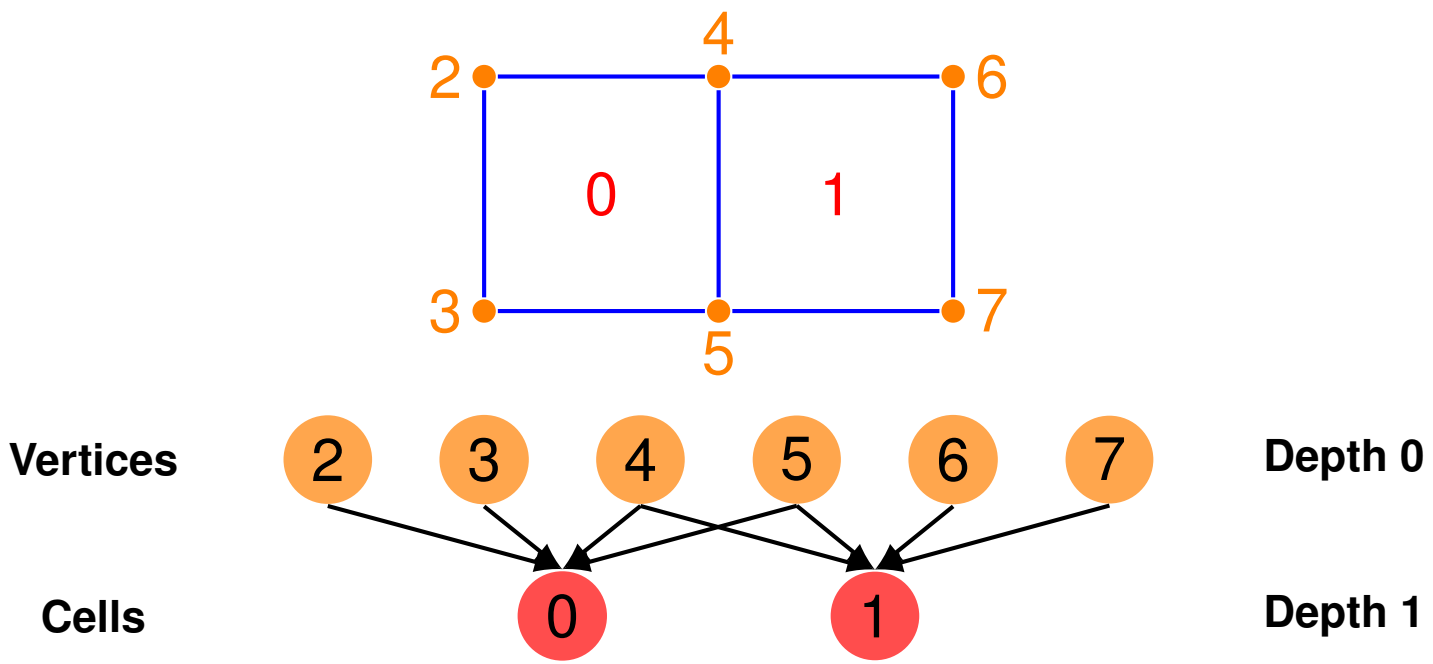
Sample Meshes

Interpolated quadrilateral mesh



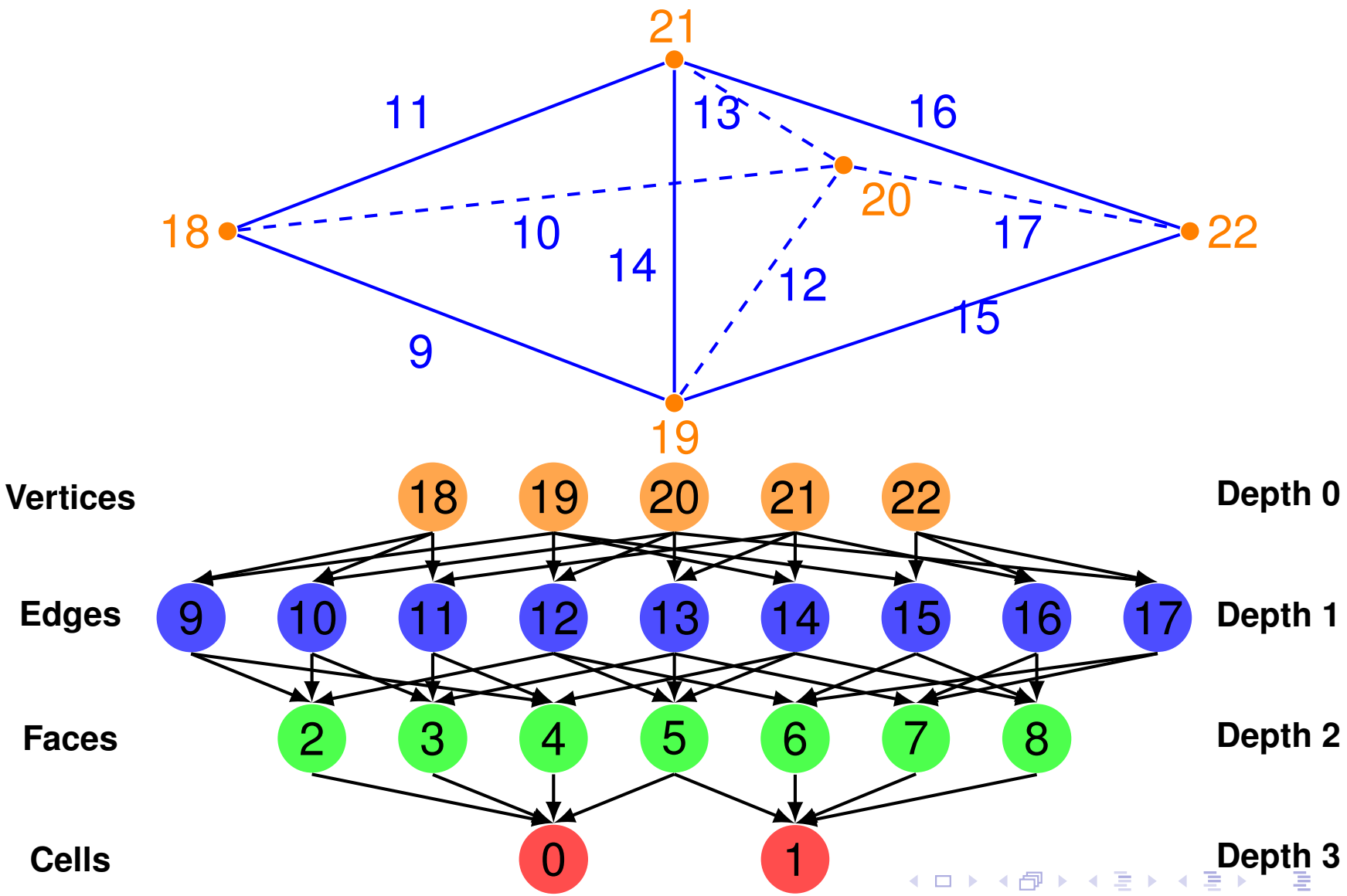
Sample Meshes

Optimized quadrilateral mesh



Sample Meshes

Interpolated tetrahedral mesh



Mesh Interface

By focusing on the key topological relations,
the interface can be both concise and quite general

- Single relation
- Dual is obtained by reversing arrows
- Can associate functions with DAG points
  - Dual operation gives the support of the function

Mesh Algorithms for PDE with Sieve I: Mesh Distribution, Sci. Prog., 2009.

New Unstructured Interface

- NO explicit references to element type
  - A point may be any mesh element
  - getCone(point): adjacent (d-1)-elements
  - getSupport(point): adjacent (d+1)-elements
- Transitive closure
  - closure(cell): The computational unit for FEM
- Algorithms independent of mesh
  - dimension
  - shape (even hybrid)
  - global topology
  - finite element

New Unstructured Interface

- NO explicit references to element type
  - A point may be any mesh element
  - getCone(point): adjacent (d-1)-elements
  - getSupport(point): adjacent (d+1)-elements
- Transitive closure
  - closure(cell): The computational unit for FEM
- Algorithms independent of mesh
  - dimension
  - shape (even hybrid)
  - global topology
  - finite element

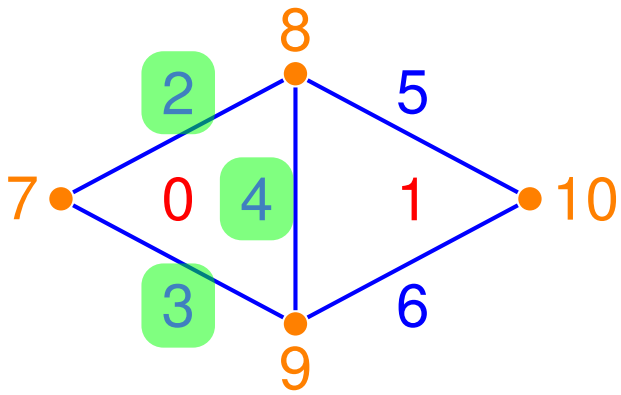
New Unstructured Interface

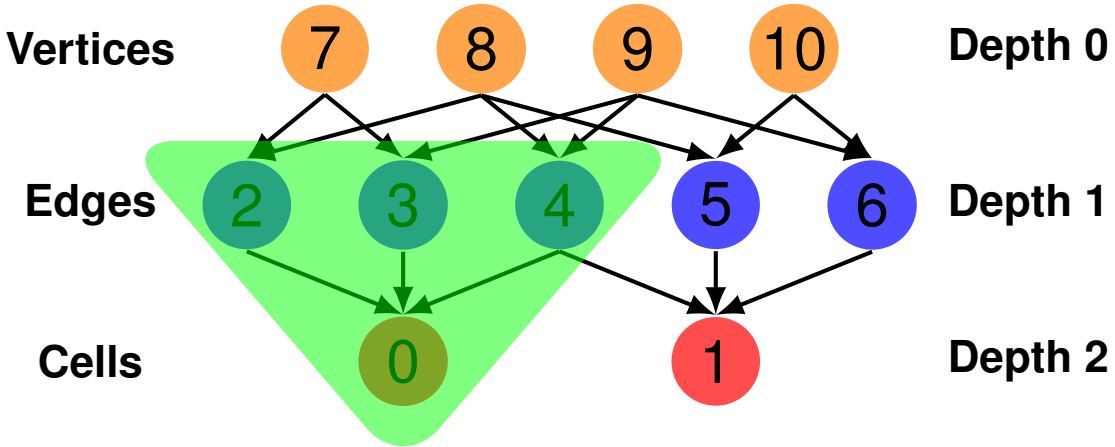
- NO explicit references to element type
  - A point may be any mesh element
  - getCone(point): adjacent (d-1)-elements
  - getSupport(point): adjacent (d+1)-elements
- Transitive closure
  - closure(cell): The computational unit for FEM
- Algorithms independent of mesh
  - dimension
  - shape (even hybrid)
  - global topology
  - finite element

Basic Operations
Cone

We begin with the basic covering relation,

cone(0) = {2, 3, 4}

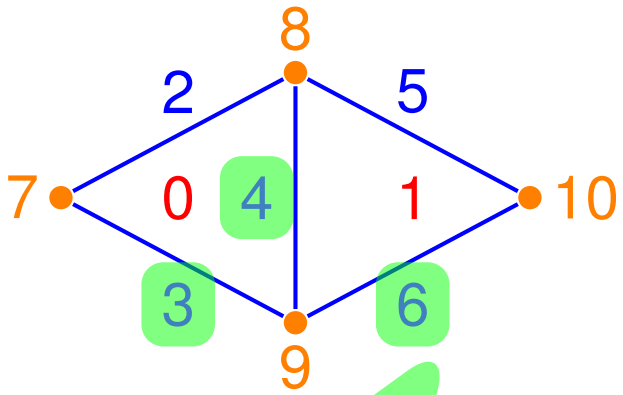


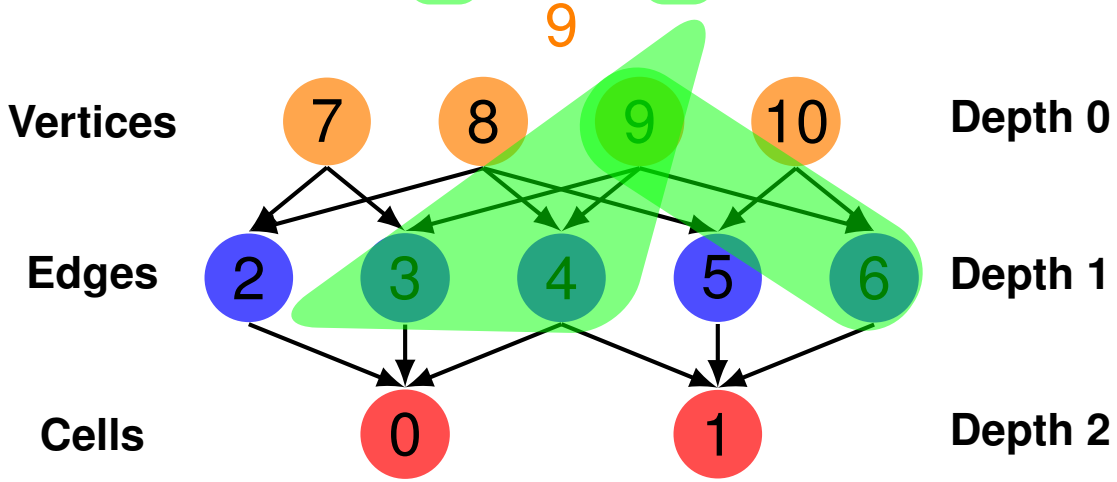


Basic Operations
Support

reverse arrows to get the dual operation,

support(9) = {3, 4, 6}

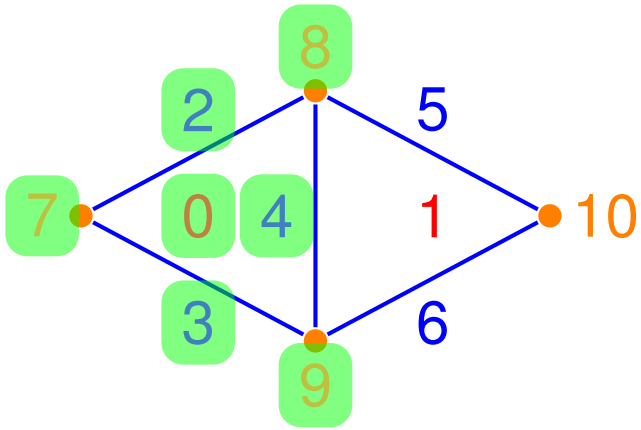


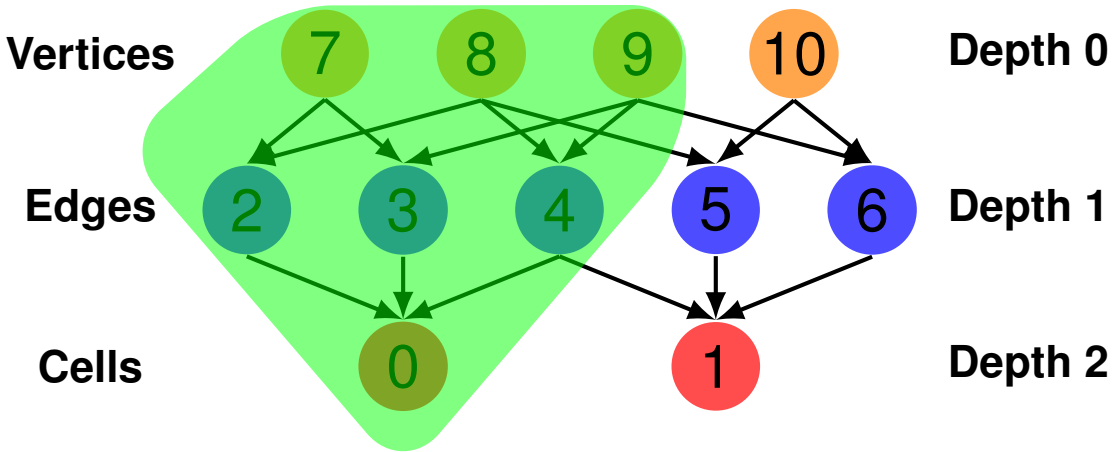


Basic Operations
Closure

add the transitive closure of the relation,

closure(0) = {0, 2, 3, 4, 7, 8, 9}

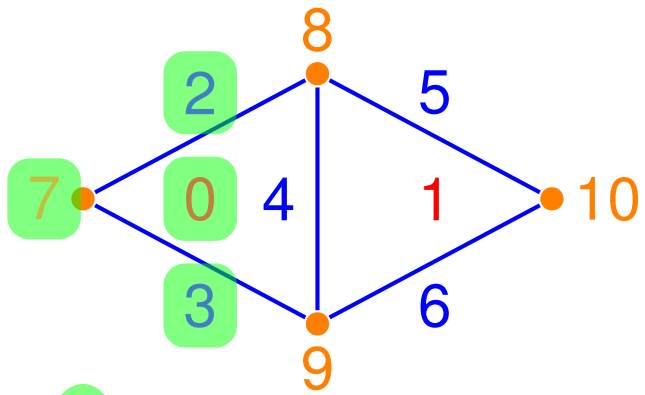


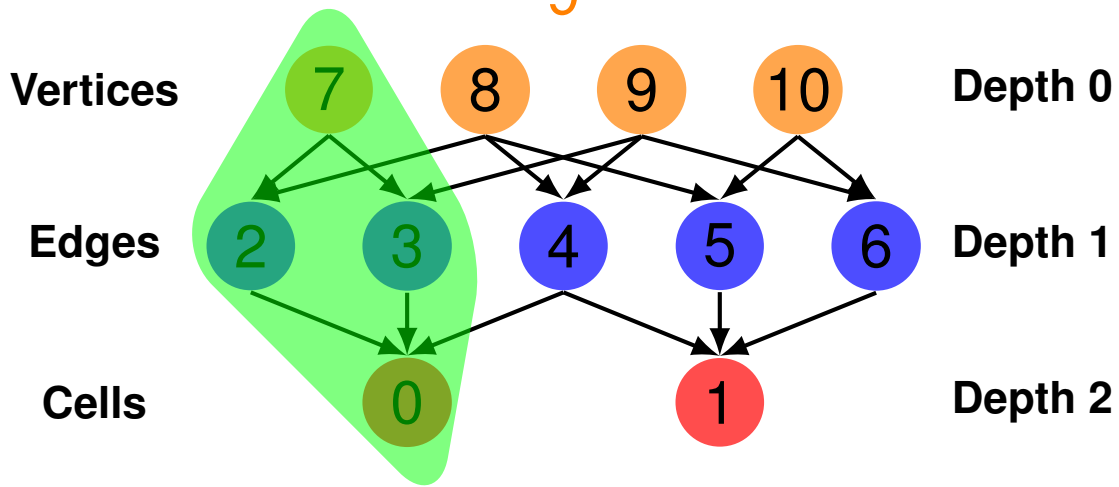


Basic Operations
Star

and the transitive closure of the dual,

star(7) = {7, 2, 3, 0}

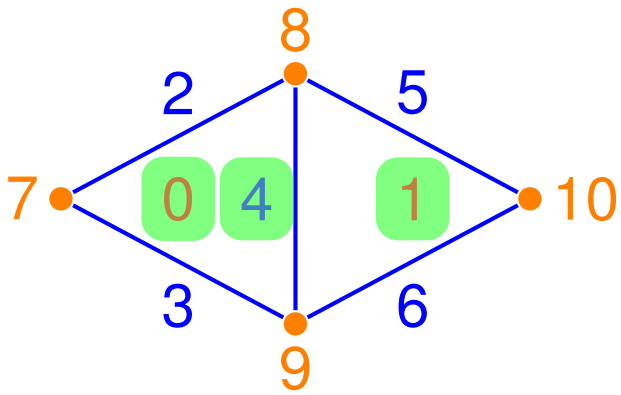


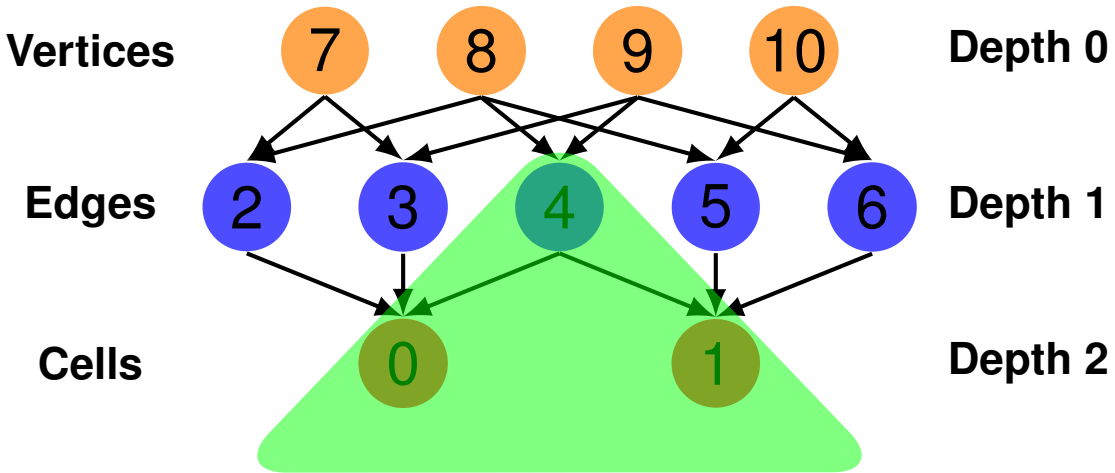


Basic Operations
Meet

and augment with lattice operations.

meet(0, 1) = {4}

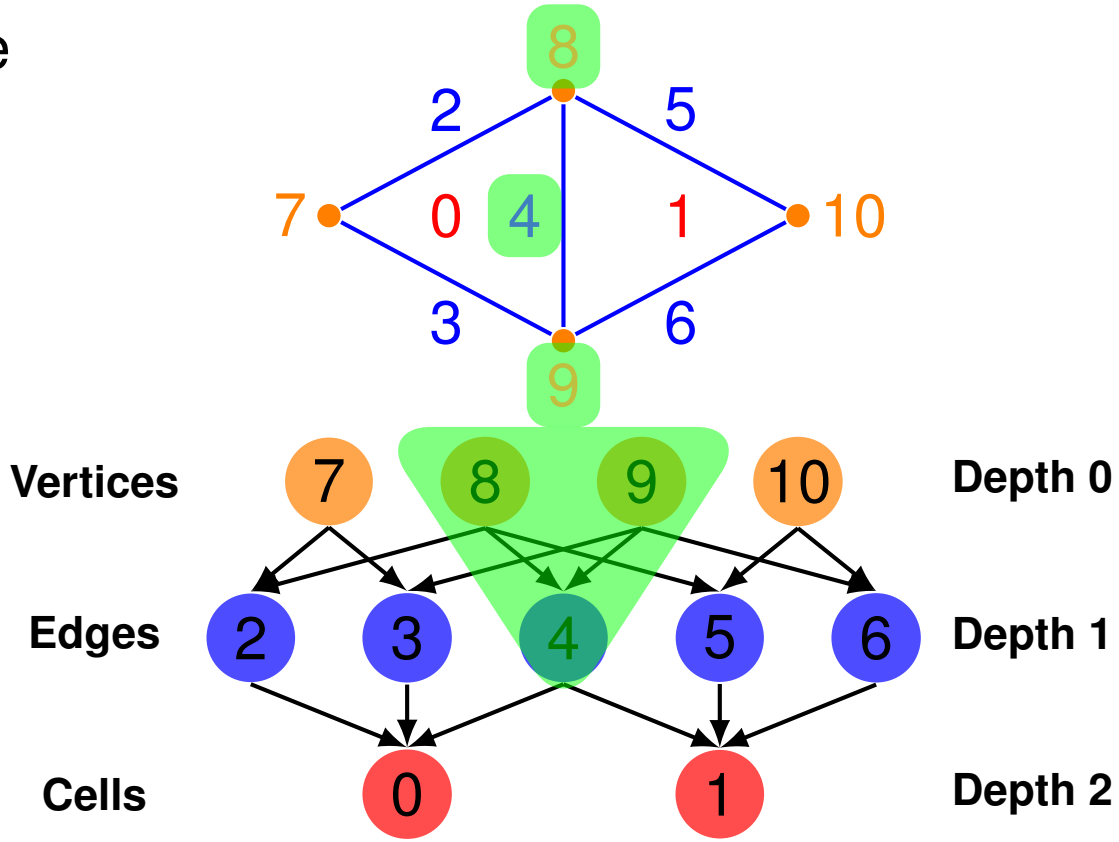




Basic Operations
Join

and augment with lattice operations.

join(8, 9) = {4}



Mesh Creation

An empty mesh can be created using

```
DMPlexCreate(MPI_Comm comm, DM *dm);
```

and then filled in using the primitives

```
DMPlexSetConeSize(DM dm, PetscInt p, PetscInt coneSize);
DMPlexSetCone(DM dm, PetscInt p, PetscInt cone[]);
```

and then

```
DMPlexSetSupportSize(DM dm, PetscInt p, PetscInt supportSize);
DMPlexSetSupport(DM dm, PetscInt p, PetscInt support[]);
```

or

```
DMPlexSymmetrize(DM dm);
```

Mesh Cloning

An existing mesh can be copied using

```
DMPlexClone(DM dm, DM *newdm);
```

so that the topology is shared.

DM structures are not shared

- Data layout (**PetscSection**)
- Communication pattern (**PetscSF**)

Mesh Input

DMPlex can input an existing mesh data using

```
DMPlexCreateFromCellList(MPI_Comm comm, PetscInt dim,
                         PetscInt numCells, PetscInt numVertices,
                         PetscInt numCorners, PetscBool interpolate,
                         const int cells[],
                         PetscInt spaceDim, const double vertexCoords[],
                         DM *dm)
```

Very common interchange format:

- Used for Triangle and TetGen
- `DMPlexCreateFromDAG()` is similar
  - single numbering
  - PETSc types

Mesh Generation

DMPlex can generate a mesh given a boundary

```
DMPlexGenerate(DM boundary, const char name[],
               PetscBool interpolate, DM *mesh);
```

which dispatches to 3rd party mesh generators.

It also has built in meshes,

- `DMPlexCreateBoxMesh()`, which calls `DMPlexGenerate()` after
  - `DMPlexCreateSquareBoundary()`
  - `DMPlexCreateCubeBoundary()`
- `DMPlexCreateHexBoxMesh()`

Mesh Refinement

DMPlex can refine a mesh using

```
DMRefine(DM dm, MPI_Comm comm, DM *refdm);
```

using 3rd party generators, or parallel uniform refinement .

We view the mesh from SNES ex62 which makes `ex62_sol.vtk`

- `./ex62 -refinement_limit 0.0625 -pc_type jacobi`
- `./ex62 -refinement_limit 0.00625 -pc_type jacobi`
- `mpiexec -n 3 ./ex62 -refinement_limit 0.00625 -dm_view_partition -pc_type jacobi -ksp_max_it 100`

where we have generated the FEM header using

```
./bin/pythonscripts/PetscGenerateFEMQuadrature.py 2 1 2 1 laplacian
2 1 1 1 gradient src/snes/examples/tutorials/ex62.h
```

Mesh Refinement

DMPlex can refine a mesh using

```
DMRefine(DM dm, MPI_Comm comm, DM *refdm);
```

using 3rd party generators, or parallel uniform refinement .

We view the mesh from SNES ex62 which makes `ex62_sol.vtk`

- `./ex62 -dim 3 -refinement_limit 0.0 -pc_type jacobi`
- `./ex62 -dim 3 -refinement_limit 0.001 -pc_type jacobi`
- `mpiexec -n 3 ./ex62 -dim 3 -refinement_limit 0.001 -dm_view_partition -pc_type jacobi -ksp_max_it 100`

where we have generated the FEM header using

```
./bin/pythonscripts/PetscGenerateFEMQuadrature.py 3 1 3 1 laplacian
3 1 1 1 gradient src/snes/examples/tutorials/ex62.h
```

Mesh Partitioning

DMPlex can partition an existing mesh

```
DMPlexCreatePartition(DM dm, PetscInt height, PetscBool enlarge,
                      PetscSection *partSection, IS *partition,
                      PetscSection *origPartSection, IS *origPartition);
```

which dispatches to 3rd party mesh partitioners.

Normally

- not called by users
- needs `DMPlexCreatePartitionClosure()`
- serial

Mesh Distribution

DMPlex can distribute an existing mesh

```
DMPlexDistribute(DM dm, const char partitioner[],
                 PetscInt overlap, DM *dmParallel)
```

- Calls `DMPlexCreatePartition()`
- Distributes coordinates and labels
- Creates **PetscSF** for the point distribution

Mesh Labels

DMLabel marks mesh points

- Markers are **PetscInt**
- Bi-directional queries
  - `DMLabelGetValue()` for a point
  - `DMLabelGetStratumIS()` for a marker
- Search optimization using `DMLabelCreateIndex()`

They can be used to:

- Define submeshes, perhaps of lower dimension
- Set material properties
- Mark ghost elements

Outline

Outline

Raw Array Access

You can get a multidimensional array from vector data:

```
DMDAVecGetArray(DM da, Vec v, void *array);
```

where the array dimension is taken from the **DM**

For a multicomponent **DM**, using

```
DMDAVecGetArrayDOF(DM da, Vec v, void *array);
```

will add one extra dimension for components.

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRows()`
- `MatZeroRowsIS()`
- `MatZeroRowsLocal()`
- `MatZeroRowsStencil()`

DM Method:

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRows()`
- `MatZeroRowsIS()`
- `MatZeroRowsLocal()`
- `MatZeroRowsStencil()`

DM Method:

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRows()`
- `MatZeroRowsIS()`
- `MatZeroRowsLocal()`
- `MatZeroRowsStencil()`

DM Method:

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRows()`
- `MatZeroRowsIS()`
- `MatZeroRowsLocal()`
- `MatZeroRowsStencil()`

DM Method:

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRows()`
- `MatZeroRowsIS()`
- `MatZeroRowsLocal()`
- `MatZeroRowsStencil()`

DM Method:

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRowsColumns()`
- `MatZeroRowsColumnsIS()`
- `MatZeroRowsColumnsLocal()`
- `MatZeroRowsColumnsStencil()`

DM Method:

Dirichlet conditions

Manual Method:

- Set rhs values to boundary values
- `MatZeroRowsColumns()`
- `MatZeroRowsColumnsIS()`
- `MatZeroRowsColumnsLocal()`
- `MatZeroRowsColumnsStencil()`

DM Method:

Check stencil for boundary points, Set rhs values to boundary values

```
/* Test whether we are on the top edge of the global array */
if (info->ys+info->ym == info->my) {
  j = info->my - 1;
  /* top edge */
  for (i = info->xs; i < info->xs+info->xm; ++i) {
    f[j][i].u     = x[j][i].u - lid;
    f[j][i].v     = x[j][i].v;
    f[j][i].omega = x[j][i].omega + (x[j][i].u - x[j-1][i].u)*dhy;
    f[j][i].temp  = x[j][i].temp-x[j-1][i].temp;
  }
}
```

Dirichlet conditions

Manual Method:

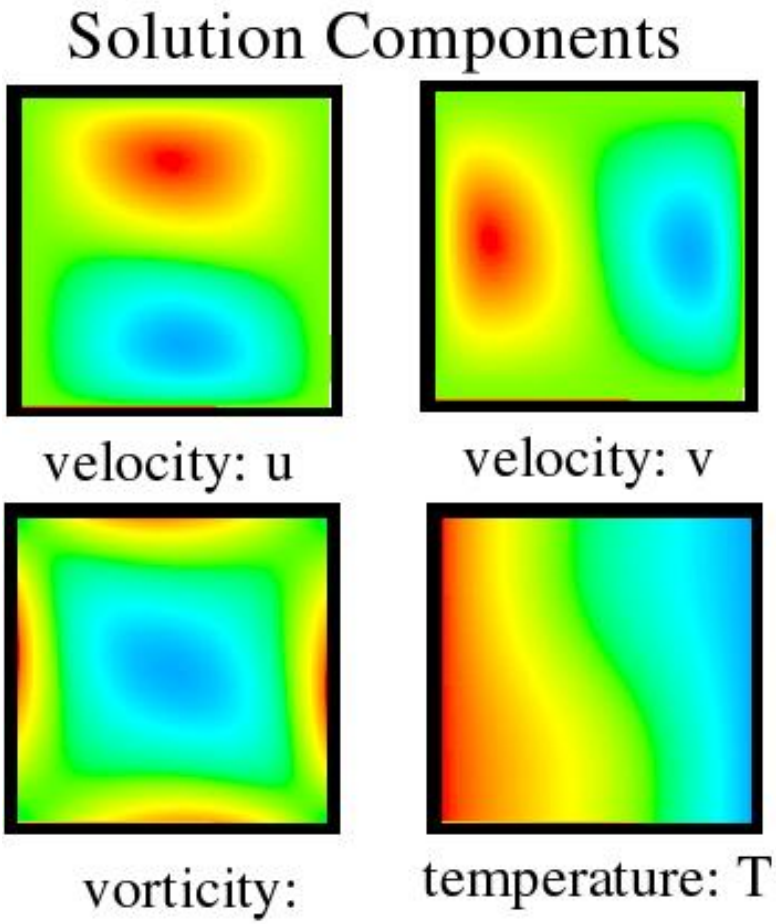
- Set rhs values to boundary values
- `MatZeroRowsColumns()`
- `MatZeroRowsColumnsIS()`
- `MatZeroRowsColumnsLocal()`
- `MatZeroRowsColumnsStencil()`

DM Method:

Check stencil for boundary points, Set Jacobian rows to the identity

```
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;
    /* boundary points */
    if (i == 0 || j == 0 || i == info->mx-1 || j == info->my-1) {
      v[0] = 2.0*(hydhx + hxdhy);
      MatSetValuesStencil(jac,1,&row,1,&row,v,INSERT_VALUES);
    }
  }
}
```

SNES Example
Driven Cavity



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

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

Driven Cavity Application Context

```
typedef struct {
  /*-----  basic application data  -----*/
  PetscReal lid_velocity;
  PetscReal prandtl
  PetscReal grashof;
  PetscBool draw_contours;
} AppCtx;
```

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

Driven Cavity Residual Evaluation

```
Residual(SNES snes, Vec X, Vec F, void *ptr) {
  AppCtx         *user = (AppCtx *) ptr;

  /* local starting and ending grid points */
  PetscInt        istart, iend, jstart, jend;
  PetscScalar    *f; /* local vector data */
  PetscReal       grashof = user->grashof;
  PetscReal       prandtl = user->prandtl;
  PetscErrorCode ierr;

  /* Code to communicate nonlocal ghost point data */
  VecGetArray(F, &f);
  /* Code to compute local function components */
  VecRestoreArray(F, &f);
  return 0;
}
```

Better Driven Cavity Residual Evaluation

```
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];
      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].u = uxx + uyy - .5*(x[j+1][i].omega-x[j-1][i].omega)*hx;
      f[j][i].v = uxx + uyy + .5*(x[j][i+1].omega-x[j][i-1].omega)*hy;
      f[j][i].omega = uxx + uyy +
          (vxp*(u - x[j][i-1].omega) + vxm*(x[j][i+1].omega - u))*hy +
          (vyp*(u - x[j-1][i].omega) + vym*(x[j+1][i].omega - u))*hx -
          0.5*grashof*(x[j][i+1].temp - x[j][i-1].temp)*hy;
      f[j][i].temp  =  uxx + uyy + prandtl*
          ((vxp*(u - x[j][i-1].temp) + vxm*(x[j][i+1].temp - u))*hy +
           (vyp*(u - x[j-1][i].temp) + vym*(x[j+1][i].temp - u))*hx);
}}}
```

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

Outline

PetscSection
What Is It?

Similar to **PetscLayout**, maps point $\longrightarrow$ (size, offset)

- Processes are replaced by points
  - Also what we might use for multicore **PetscLayout**
- Boundary conditions are just another **PetscSection**
  - Map points to number of constrained dofs
  - Offsets into integer array of constrained local dofs
- Fields are just another **PetscSection**
  - Map points to number of field dofs
  - Offsets into array with all fields
- Usable by all **DM** subclasses
  - Structured grids with **DMDA**
  - Unstructured grids with **DMPlex**

PetscSection
Why Use It?

PETSc Solvers only understand Integers

Decouples Mesh From Discretization

- Mesh does not need to know how dofs are generated, just how many are attached to each point.
- It does not matter whether you use FD, FVM, FEM, etc.

Decouples Mesh from Solver

- Solver gets the data layout and partitioning from **Vec** and **Mat**, nothing else from the mesh.
- Solver gets restriction/interpolation matrices from **DM**.

Decouples Discretization from Solver

- Solver only gets the field division, nothing else from discretization.

PetscSection
How Do I Build One?

High Level Interface

```
DMComplexCreateSection(
    DM dm, PetscInt dim, PetscInt numFields,
      PetscInt numComp[], PetscInt numDof[],
    PetscInt numBC, PetscInt bcField[], IS bcPoints[],
    PetscSection *section);
```

Discretization	Dof/Dimension
$P_1 - P_0$	[2 0 0 0 | 0 0 0 1]
$Q_2 - Q_1$	[2 2 0 0 | 1 0 0 0]
$Q_2 - P_1^{\text{disc}}$	[2 2 0 0 | 0 0 0 3]

Data Layout

PetscSection defines a data layout

- maps $p \rightarrow (off, off + 1, \ldots, off + dof)$
- where $p \in [pStart, pEnd)$, called the *chart*
- ranges can be divided into parts, called *fields*
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- `PetscSectionGetOffset()`, `PetscSectionGetDof()`
- where $p \in [pStart, pEnd)$, called the *chart*
- ranges can be divided into parts, called *fields*
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- maps $p \rightarrow (off, off + 1, \ldots, off + dof)$
- where $p \in [pStart, pEnd)$, called the *chart*
- ranges can be divided into parts, called *fields*
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- maps $p \rightarrow (off, off + 1, \ldots, off + dof)$
- `PetscSectionGetChart()`
- ranges can be divided into parts, called *fields*
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- maps $p \rightarrow (off, off + 1, \ldots, off + dof)$
- where $p \in [pStart, pEnd)$, called the *chart*
- ranges can be divided into parts, called *fields*
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- maps $p \to (off, off + 1, \ldots, off + dof)$
- where $p \in [pStart, pEnd)$, called the *chart*
- `PetscSectionGetFieldOffset()`, `PetscSectionGetFieldDof()`
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- maps $p \rightarrow (off, off + 1, \ldots, off + dof)$
- where $p \in [pStart, pEnd)$, called the *chart*
- ranges can be divided into parts, called *fields*
- prefix sums calculated automatically on setup

Data Layout

PetscSection defines a data layout

- maps $p \rightarrow (off, off + 1, \ldots, off + dof)$
- where $p \in [pStart, pEnd)$, called the *chart*
- ranges can be divided into parts, called *fields*
- `PetscSectionSetUp()`

Using PetscSection

PetscSection can be used to segment data

- Use **Vec** and **IS** to store data
- Use point *p* instead of index *i*
- Maps to a set of values instead of just one

We provide a convenience method for extraction

```
VecGetValuesSection(Vec v, PetscSection s, PetscInt p, PetscScalar **a);
```

which works in an analogous way to

```
MatSetValuesStencil(Mat A, PetscInt nr, const MatStencil rs[],
                    PetscInt nc, const MatStencil cs[],
                    const PetscScalar v[], InsertMode m);
```

Using PetscSection

PetscSection can be used to segment data

- Use **Vec** and **IS** to store data
- Use point *p* instead of index *i*
- Maps to a set of values instead of just one

We can get the layout of coordinates over the mesh

```
DMPlexGetCoordinateSection(DM dm, PetscSection *s);
```

where the data is stored in a **Vec**

```
DMGetCoordinates(DM dm, Vec *coords);
```

Using PetscSection

PetscSection can be used to segment data

- Use **Vec** and **IS** to store data
- Use point *p* instead of index *i*
- Maps to a set of values instead of just one

We can retrieve FEM data from vector without complicated indexing,

```
DMPlexVecGetClosure(DM dm, PetscSection s, Vec v,
                    PetscInt cell, PetscInt *, PetscScalar *a[]);
```

and the same thing works for matrices

```
DMPlexMatSetClosure(DM dm, PetscSection rs, PetscSection cs, Mat A,
                    PetscInt p, const PetscScalar v[], InsertMode m);
```

Constraints

PetscSection allows some dofs to be marked as *constrained*:

- specify the number of constraints for each point,
- and their pointwise offsets.
- typically used for Dirichlet conditions

Constraints

PetscSection allows some dofs to be marked as *constrained*:

- `PetscSectionGetConstraintDof()`, `PetscSectionGetFieldConstraintDof()`
- and their pointwise offsets.
- typically used for Dirichlet conditions

Constraints

PetscSection allows some dofs to be marked as *constrained*:

- specify the number of constraints for each point,
- and their pointwise offsets.
- typically used for Dirichlet conditions

Constraints

PetscSection allows some dofs to be marked as *constrained*:

- specify the number of constraints for each point,
- `PetscSectionGetConstraintIndices()`, `PetscSectionGetFieldConstraintIndices()`
- typically used for Dirichlet conditions

Constraints

PetscSection allows some dofs to be marked as *constrained*:

- specify the number of constraints for each point,
- and their pointwise offsets.
- typically used for Dirichlet conditions

Constraints

PetscSection allows some dofs to be marked as *constrained*:

- specify the number of constraints for each point,
- and their pointwise offsets.
- `PetscSectionCreateGlobalSection()`

Global Sections

A *global* section

- has no constrained dofs
- has only shared dofs which are owned
- is layout for **DM** global vectors

and can be created using

```
PetscSectionCreateGlobalSection(PetscSection s, PetscSF pointSF,
                                PetscBool includeConstraints,
                                PetscSection *gs);
```

Interaction with PetscSF

We use **PetscSF** to describe shared points

Composing a point **PetscSF** and **PetscSection**, we can build

- a global section
- a **PetscSF** for shared dofs

This *composability* means we can build hierarchies of sections and pieces of sections.

Interaction with PetscSF

We use **PetscSF** to describe shared points

Composing a point **PetscSF** and **PetscSection**, we can build

- `PetscSectionCreateGlobalSection()`
- a **PetscSF** for shared dofs

This *composability* means we can build hierarchies of sections and pieces of sections.

Interaction with PetscSF

We use **PetscSF** to describe shared points

Composing a point **PetscSF** and **PetscSection**, we can build

- a global section
- a **PetscSF** for shared dofs

This *composability* means we can build hierarchies of sections and pieces of sections.

Interaction with PetscSF

We use **PetscSF** to describe shared points

Composing a point **PetscSF** and **PetscSection**, we can build

- a global section
- `PetscSFCreateSectionSF()`

This *composability* means we can build hierarchies of sections and pieces of sections.

Subsections

A **PetscSection** can also be broken apart to represent smaller pieces of the problem for

- subsolves
- output
- postprocessing

A subsection can be extracted

- from a subset of fields
- from a subset of points

Subsections

A **PetscSection** can also be broken apart to represent smaller pieces of the problem for

- subsolves
- output
- postprocessing

A subsection can be extracted

- `PetscSectionCreateSubsection()`
- from a subset of points

Subsections

A **PetscSection** can also be broken apart to represent smaller pieces of the problem for

- subsolves
- output
- postprocessing

A subsection can be extracted

- from a subset of fields
- from a subset of points

Subsections

A **PetscSection** can also be broken apart to represent smaller pieces of the problem for

- subsolves
- output
- postprocessing

A subsection can be extracted

- from a subset of fields
- `PetscSectionCreateSubmeshSection()`

Residual Evaluation

I developed a single residual evaluation routine independent of spatial dimension, cell geometry, and finite element:

$$F(\vec{u}) = 0$$

Dim	Cell Types	Discretizations
1	Simplex	Lagrange FEM
2	Tensor Product	H(div) FEM*
3	Polyhedral	H(curl) FEM*
6†	Prism	DG FEM *‡

† Peter Brune, ANL

* FEniCS Project

‡ Blaise Bourdin, LSU

We have also implemented a polyhedral FVM,
but this required changes to the residual evaluation.

Residual Evaluation

I developed a single residual evaluation routine independent of spatial dimension, cell geometry, and finite element:

$$F(\vec{u}) = 0$$

Dim	Cell Types	Discretizations
1	Simplex	Lagrange FEM
2	Tensor Product	H(div) FEM*
3	Polyhedral	H(curl) FEM*
6†	Prism	DG FEM *‡

† Peter Brune, ANL

* FEniCS Project

‡ Blaise Bourdin, LSU

We have also implemented a polyhedral FVM,
but this required changes to the residual evaluation.

Residual Evaluation

I developed a single residual evaluation routine independent of spatial dimension, cell geometry, and finite element:

$$F(\vec{u}) = 0$$

Dim
1
2
3
6†

Cell Types
Simplex
Tensor Product
Polyhedral
Prism

Discretizations
Lagrange FEM
H(div) FEM*
H(curl) FEM*
DG FEM *‡

† Peter Brune, ANL

* FEniCS Project

‡ Blaise Bourdin, LSU

We have also implemented a polyhedral FVM, but this required changes to the residual evaluation.

Residual Evaluation

I developed a single residual evaluation routine independent of spatial dimension, cell geometry, and finite element:

$$F(\vec{u}) = 0$$

Dim
1
2
3
6†

Cell Types
Simplex
Tensor Product
Polyhedral
Prism

Discretizations
Lagrange FEM
H(div) FEM*
H(curl) FEM*
DG FEM *‡

† Peter Brune, ANL

* FEniCS Project

‡ Blaise Bourdin, LSU

We have also implemented a polyhedral FVM, but this required changes to the residual evaluation.

Residual Evaluation

I developed a single residual evaluation routine independent of spatial dimension, cell geometry, and finite element:

$$F(\vec{u}) = 0$$

Dim
1
2
3
6†

Cell Types
Simplex
Tensor Product
Polyhedral
Prism

Discretizations
Lagrange FEM
H(div) FEM*
H(curl) FEM*
DG FEM *‡

† Peter Brune, ANL

* FEniCS Project

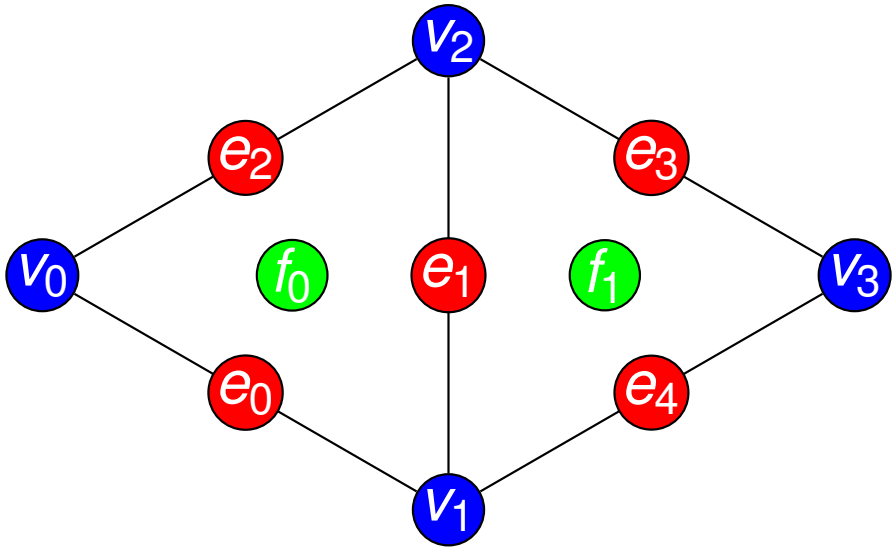
‡ Blaise Bourdin, LSU

We have also implemented a polyhedral FVM,
but this required changes to the residual evaluation.

Outline

DMComplex
SNES ex62

$P_2 - P_1$ Stokes Example

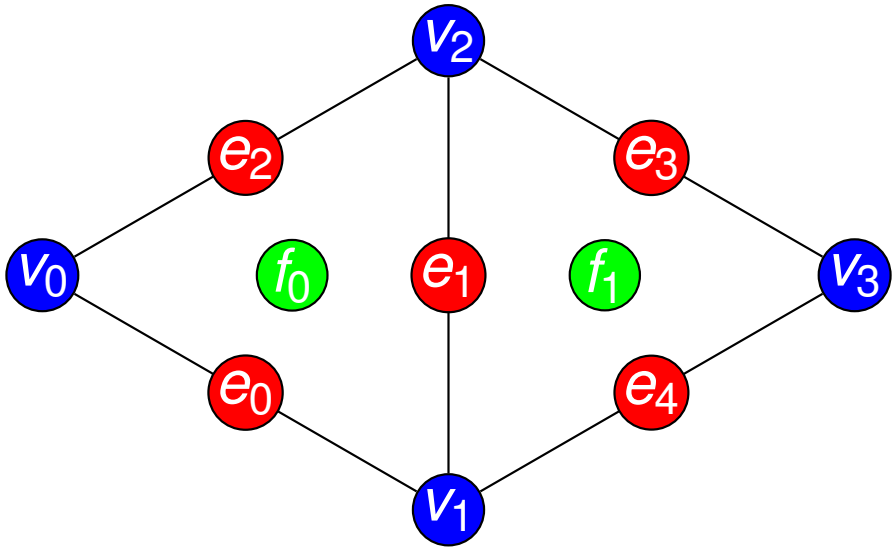


Naively, we have

$$\begin{aligned} \text{cl}(\text{cell}) = \quad & [fe_0\, e_1\, e_2\, v_0\, v_1\, v_2 \qquad\qquad\qquad\qquad\qquad\qquad\quad] \\ x(\text{cell}) = \quad & [u_{e_0}\, v_{e_0}\, u_{e_1}\, v_{e_1}\, u_{e_2}\, v_{e_2} \\ & \quad u_{v_0}\, v_{v_0}\, p_{v_0}\, u_{v_1}\, v_{v_1}\, p_{v_1}\, u_{v_2}\, v_{v_2}\, p_{v_2} \quad] \end{aligned}$$

DMComplex
SNES ex62

$P_2 - P_1$ Stokes Example



We reorder so that fields are contiguous

$$x'(\text{cell}) = [u_{e_0} v_{e_0} u_{e_1} v_{e_1} u_{e_2} v_{e_2} \; u_{v_0} v_{v_0} u_{v_1} v_{v_1} u_{v_2} v_{v_2} \; p_{v_0} p_{v_1} p_{v_2}]$$

FEM Integration Model

Proposed by Jed Brown

We consider weak forms dependent only on fields and gradients,

$$\int_\Omega \phi \cdot f_0(u, \nabla u) + \nabla\phi : \vec{f}_1(u, \nabla u) = 0. \tag{1}$$

Discretizing we have

$$\sum_e \mathcal{E}_e^T \left[B^T W^q f_0(u^q, \nabla u^q) + \sum_k D_k^T W^q \vec{f}_1^k(u^q, \nabla u^q) \right] = 0 \tag{2}$$

f_n	pointwise physics functions
u^q	field at a quad point
W^q	diagonal matrix of quad weights
B,D	basis function matrices which reduce over quad points
$\mathcal{E}$	assembly operator

Batch Integration

```
DMPlexComputeResidualFEM(dm, X, F, user)
{
  VecSet(F, 0.0);
  <Put boundary conditions into local input vector>
  <Extract coefficients and geometry for batch>
  <Integrate batch of elements>
  <Insert batch of element vectors into global vector>
}
```

Batch Integration
Set boundary conditions

```
DMPlexComputeResidualFEM(dm, X, F, user)
{
  VecSet(F, 0.0);
  DMPlexProjectFunctionLocal(dm, numComponents,
    bcFuncs, INSERT_BC_VALUES, X);
  <Extract coefficients and geometry for batch>
  <Integrate batch of elements>
  <Insert batch of element vectors into global vector>
}
```

Batch Integration

Extract coefficients and geometry

```
DMPlexComputeResidualFEM(dm, X, F, user)
{
  VecSet(F, 0.0);
  <Put boundary conditions into local input vector>
  DMPlexGetHeightStratum(dm, 0, &cStart, &cEnd);
  for (c = cStart; c < cEnd; ++c) {
    DMPlexComputeCellGeometry(dm, c, &v0[c*dim],
      &J[c*dim*dim], &invJ[c*dim*dim], &detJ[c]);
    DMPlexVecGetClosure(dm, NULL, X, c, NULL, &x);
    for (i = 0; i < cellDof; ++i) u[c*cellDof+i] = x[i];
    DMPlexVecRestoreClosure(dm, NULL, X, c, NULL, &x);
  }
  <Integrate batch of elements>
  <Insert batch of element vectors into global vector>
}
```

Batch Integration

Integrate element batch

```
DMPlexComputeResidualFEM(dm, X, F, user)
{
  VecSet(F, 0.0);
  <Put boundary conditions into local input vector>
  <Extract coefficients and geometry for batch>
  for (field = 0; field < numFields; ++field) {
    (*mesh->integrateResidualFEM)(Ne, numFields, field,
      quad, u,
      v0, J, invJ, detJ,
      f0, f1, elemVec);
    (*mesh->integrateResidualFEM)(Nr, ...);
  }
  <Insert batch of element vectors into global vector>
}
```

Batch Integration

Insert element vectors

```
DMPlexComputeResidualFEM(dm, X, F, user)
{
  VecSet(F, 0.0);
  <Put boundary conditions into local input vector>
  <Extract coefficients and geometry for batch>
  <Integrate batch of elements>
  for (c = cStart; c < cEnd; ++c) {
    DMPlexVecSetClosure(dm, NULL, F, c,
      &elemVec[c*cellDof], ADD_VALUES);
  }
}
```

Element Integration

```
FEMIntegrateResidualBatch(Ne, numFields, field,
  quad[], coefficients[],
  v0s[], jacobians[], jacobianInv[], jacobianDet[],
  f0_func, f1_func)
{
  <Loop over batch of elements (e)>
    <Loop over quadrature points (q)>
      <Make x_q>
      <Make u_q and gradU_q>
      <Call f_0 and f_1>
    <Loop over element vector entries (f, fc)>
      <Add contributions from f_0 and f_1>
}
```

Element Integration

Calculate x_q

```
FEMIntegrateResidualBatch(...)
{
  <Loop over batch of elements (e)>
    <Loop over quadrature points (q)>
      for (d = 0; d < dim; ++d) {
        x[d] = v0[d];
        for (d2 = 0; d2 < dim; ++d2) {
          x[d] += J[d*dim+d2]*(quadPoints[q*dim+d2]+1);
        }
      }
      <Make x_q>
      <Make u_q and gradU_q>
      <Call f_0 and f_1>
    <Loop over element vector entries (f, fc)>
      <Add contributions from f_0 and f_1>
}
```

Element Integration

Calculate u_q and ∇u_q

```
FEMIntegrateResidualBatch(...)
{
  <Loop over batch of elements (e)>
    <Loop over quadrature points (q)>
      <Make x_q>
      for (f = 0; f < numFields; ++f) {
        for (b = 0; b < Nb; ++b) {
          for (comp = 0; comp < Ncomp; ++comp) {
            u[comp] += coefficients[cidx]*basis[q+cidx];
            for (d = 0; d < dim; ++d) {
              <Transform derivative to real space>
              gradU[comp*dim+d] +=
                coefficients[cidx]*realSpaceDer[d];
            }
          }
        }
      }
      <Call f_0 and f_1>
    <Loop over element vector entries (f, fc)>
```

Element Integration

Calculate u_q and ∇u_q

```
FEMIntegrateResidualBatch(...)
{
  <Loop over batch of elements (e)>
    <Loop over quadrature points (q)>
      <Make x_q>
      for (f = 0; f < numFields; ++f) {
        for (b = 0; b < Nb; ++b) {
          for (comp = 0; comp < Ncomp; ++comp) {
            u[comp] += coefficients[cidx]*basis[q+cidx];
            for (d = 0; d < dim; ++d) {
              realSpaceDer[d] = 0.0;
              for (g = 0; g < dim; ++g) {
                realSpaceDer[d] +=
                  invJ[g*dim+d]*basisDer[(q+cidx)*dim+g];
              }
              gradU[comp*dim+d] +=
                coefficients[cidx]*realSpaceDer[d];
            }
          }
```

Element Integration

Call f_0 and f_1

```
FEMIntegrateResidualBatch(...)
{
  <Loop over batch of elements (e)>
    <Loop over quadrature points (q)>
      <Make x_q>
      <Make u_q and gradU_q>
      f0_func(u, gradU, x, &f0[q*Ncomp]);
      for (i = 0; i < Ncomp; ++i) {
        f0[q*Ncomp+i] *= detJ*quadWeights[q];
      }
      f1_func(u, gradU, x, &f1[q*Ncomp*dim]);
      for (i = 0; i < Ncomp*dim; ++i) {
        f1[q*Ncomp*dim+i] *= detJ*quadWeights[q];
      }
    <Loop over element vector entries (f, fc)>
      <Add contributions from f_0 and f_1>
}
```

Element Integration

Update element vector

```
FEMIntegrateResidualBatch(...)
{
  <Loop over batch of elements (e)>
    <Loop over quadrature points (q)>
      <Make x_q>
      <Make u_q and gradU_q>
      <Call f_0 and f_1>
    <Loop over element vector entries (f, fc)>
      for (q = 0; q < Nq; ++q) {
        elemVec[cidx] += basis[q+cidx]*f0[q+comp];
        for (d = 0; d < dim; ++d) {
          <Transform derivative to real space>
          elemVec[cidx] +=
            realSpaceDer[d]*f1[(q+comp)*dim+d];
        }
      }
}
```

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- `PetscQuadrature` and `PetscFEM` structs
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- `DMPlexSetFEMIntegration()`
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- `DMPlexComputeResidualFEM()`
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- `DMPlexComputeJacobianFEM()` and `DMPlexComputeJacobianActionFEM()`
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- `DMPlexComputeL2Diff()`
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- Compute an L_2 projection into the element space

FEM Infrastructure

DMPlex provides support for FEM in the Brown Model:

- Specify f_0 and $\vec{f}_1$, a quadrature rule, and an element tabulation
- Specify integration methods for single field element batches
  - Have initial implementations for CPU and GPU
- Compute a parallel, multifield residual
- Compute a parallel, multifield Jacobian
- Compute an L_2 norm
- `DMPlexProjectFunction()`

FEM Geometry

The FEM infrastructure depends on `DMPlexComputeCellGeometry()`

Quantity	Description
v0	translation part of the affine map
J	Jacobian of the map from the reference element
invJ	inverse of the Jacobian
detJ	Jacobian determinant

It is likely that we will expand this set of geometric quantities.

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

http://fenicsproject.org/

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

Can build arbitrary elements by specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

http://fenicsproject.org/

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

Can build arbitrary elements by specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project

Condition of the Laplacian

2D P_1 Lagrange Elements

Num. Elements	Longest edge (h)	κ	L_2 error
64	1/4	12.6	0.0174
128	$\sqrt{2}/8$	25.2	0.00607
256	1/8	51.5	0.00434
512	$\sqrt{2}/16$	103.1	0.00153
256	1/16	207.2	0.00109
1024	$\sqrt{2}/32$	414.3	0.000381
2048	1/32	829.7	0.000271
4096	$\sqrt{2}/64$	1659.4	0.0000952
8192	1/64	3319.8	0.0000678

so we have

$$\kappa \approx 0.8h^{-2} \tag{3}$$

Condition of the Laplacian

2D P_2 Lagrange Elements

Num. Elements	Longest edge (h)	κ	L_2 error
64	1/4	68.1	2.73e-11
128	$\sqrt{2}/8$	137.2	1.64e-10
256	1/8	275.6	1.04e-09
512	$\sqrt{2}/16$	552.2	7.74e-10
256	1/16	1105.6	3.26e-09
1024	$\sqrt{2}/32$	2212.3	3.22e-09
2048	1/32	4425.7	1.02e-08
4096	$\sqrt{2}/64$	8852.6	1.13e-08
8192	1/64	17708.1	3.19e-08

so we have

$$\kappa \approx 4.3h^{-2} \tag{4}$$

The Stokes Problem – Strong Form

$$
\begin{aligned}
-\Delta u + \nabla p &= f \\
\nabla \cdot u &= 0 \\
u|_{\partial\Omega} &= g \\
\int_\Omega p &= 0
\end{aligned}
$$

The Stokes Problem – Weak Form

For $u, v \in V$ and $p, q \in \Pi$

$$< \nabla v, \nabla u > - < \nabla \cdot v, p > = < v, f >$$
$$< q, \nabla \cdot u > = 0$$
$$u|_{\partial\Omega} = g$$
$$\int_\Omega p = 0$$

2D Exact Solution

$$u = x^2 + y^2$$
$$v = 2x^2 - 2xy$$
$$p = x + y - 1$$
$$f_i = 3$$

3D Exact Solution

$$u = x^2 + y^2$$
$$v = y^2 + z^2$$
$$w = x^2 + y^2 - 2(x + y)z$$
$$p = x + y + z - 3/2$$
$$f_i = 3$$

Condition of the Stokes Operator

2D P_2/P_1 Lagrange Elements

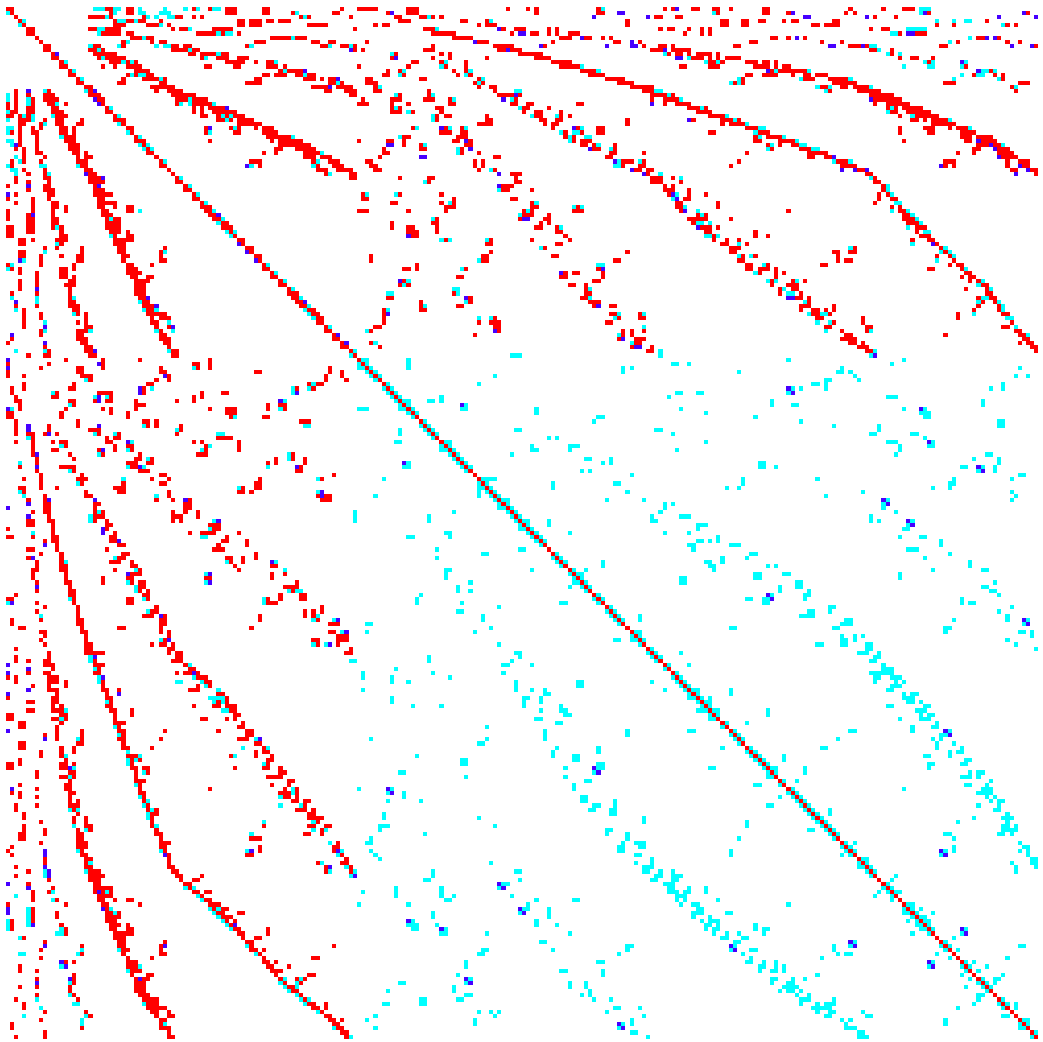
Num. Elements	Longest edge (h)	κ	L_2 error
64	1/4	7909	6.96e-07
128	$\sqrt{2}/8$	29522	1.41e-07
256	1/8	32300	7.82e-07
512	$\sqrt{2}/16$	119053	1.27e-06
256	1/16	129883	2.28e-06
1024	$\sqrt{2}/32$	466023	4.99e-06
2048	1/32	520163	6.66e-06
4096	$\sqrt{2}/64$	1121260	2.97e-05
8192	1/64	2075950	1.97e-05

so we have

$$\kappa \approx 700 h^{-2} \tag{5}$$

Jacobian

P_2/P_1 elements



Outline

FVM Geometry

The FVM infrastructure depends on `DMPlexComputeCellGeometryFVM()`, which computes

Quantity	Description
vol	cell volume or face area
centroid	cell or face centroid
normal	face normal or 0

Second Order TVD Finite Volume Method

Physics

TS ex11.c

- Advection
- Shallow Water
- Euler

Second Order TVD Finite Volume Method

Limiters

TS ex11.c

- Minmod
- van Leer
- van Albada
- Sin
- Superbee
- MC (Barth-Jespersen)

Second Order TVD Finite Volume Method

Physics Creation

```
PetscErrorCode PhysicsCreate_Advect(Model mod, Physics phys)
{
  Physics_Advect *advect = (Physics_Advect *) phys->data;
  const PetscInt inflowids[] = {100,200,300},outflowids[] = {101};
  phys->field_desc = PhysicsFields_Advect;
  phys->riemann    = PhysicsRiemann_Advect;
  /* Register "canned" boundary conditions and defaults ids */
  ModelBoundaryRegister(mod, "inflow", PhysicsBoundary_Advect_Inflow,
                        phys, ALEN(inflowids), inflowids);
  ModelBoundaryRegister(mod, "outflow", PhysicsBoundary_Advect_Outflow,
                        phys, ALEN(outflowids), outflowids);
  /* Initial/transient solution with default boundary conditions */
  ModelSolutionSetDefault(mod, PhysicsSolution_Advect, phys);
  /* Register "canned" functionals */
  ModelFunctionalRegister(mod, "Error", &advect->functional.Error,
                          PhysicsFunctional_Advect, phys);
}
```

Second Order TVD Finite Volume Method

Physics Creation

```
static PetscErrorCode PhysicsCreate_SW(Model mod,Physics phys)
{
  Physics_SW *sw = (Physics_SW *) phys->data;
  const PetscInt wallids[] = {100,101,200,300};
  phys->field_desc = PhysicsFields_SW;
  phys->riemann    = PhysicsRiemann_SW;
  phys->maxspeed   = PetscSqrtReal(2.0*sw->gravity); /* Mach 1 at depth 2
  ModelBoundaryRegister(mod, "wall", PhysicsBoundary_SW_Wall,
                        phys, ALEN(wallids), wallids);
  ModelSolutionSetDefault(mod, PhysicsSolution_SW, phys);
  ModelFunctionalRegister(mod, "Height", &sw->functional.Height,
                          PhysicsFunctional_SW, phys);
  ModelFunctionalRegister(mod, "Speed", &sw->functional.Speed,
                          PhysicsFunctional_SW, phys);
  ModelFunctionalRegister(mod, "Energy", &sw->functional.Energy,
                          PhysicsFunctional_SW, phys);
}
```

Second Order TVD Finite Volume Method

Physics Creation

```
PetscErrorCode PhysicsCreate_Euler(Model mod, Physics phys)
{
  PhysicsEuler *eu = (PhysicsEuler *) phys->data;
  const PetscInt wallids[] = {100,101,200,300};
  phys->field_desc = PhysicsFields_Euler;
  phys->riemann    = PhysicsRiemann_Euler_Rusanov;
  phys->maxspeed   = 1.0;
  ModelBoundaryRegister(mod, "wall", PhysicsBoundary_Euler_Wall,
                        phys, ALEN(wallids), wallids);
  ModelSolutionSetDefault(mod, PhysicsSolution_Euler, phys);
  ModelFunctionalRegister(mod, "Speed", &eu->monitor.Speed,
                          PhysicsFunctional_Euler, phys);
  ModelFunctionalRegister(mod, "Energy", &eu->monitor.Energy,
                          PhysicsFunctional_Euler, phys);
  ModelFunctionalRegister(mod, "Density", &eu->monitor.Density,
                          PhysicsFunctional_Euler, phys);
  ModelFunctionalRegister(mod, "Momentum", &eu->monitor.Momentum,
                          PhysicsFunctional_Euler, phys);
  ModelFunctionalRegister(mod, "Pressure", &eu->monitor.Pressure,
                          PhysicsFunctional_Euler, phys);
}
```

Second Order TVD Finite Volume Method

Residual

We begin by localizing and applying boundary conditions:

```
RHSFunction(TS ts, PetscReal time, Vec X, Vec F, void *ctx) {
  TSGetDM(ts, &dm);
  DMGetLocalVector(dm, &locX);
  DMGlobalToLocalBegin(dm, X, INSERT_VALUES, locX);
  DMGlobalToLocalEnd(dm, X, INSERT_VALUES, locX);
  ApplyBC(dm, time, locX, user);
  VecZeroEntries(F);
  (*user->RHSFunctionLocal)(dm, dmFace, dmCell, time, locX, F, user);
  DMRestoreLocalVector(dm, &locX);
}
```

Second Order TVD Finite Volume Method

Residual

By default, we call the Riemann solver for local faces:

```
RHSFunctionLocal_Upwind(DM dm, DM dmFace, DM dmCell, PetscReal time,
                        Vec locX, Vec F, User user) {
  DMPlexGetHeightStratum(dm, 1, &fStart, &fEnd);
  for (face = fStart; face < fEnd; ++face) {
    DMPlexGetLabelValue(dm, "ghost", face, &ghost);
    if (ghost >= 0) continue;
    DMPlexGetSupport(dm, face, &cells);
    DMPlexPointLocalRead(dmFace, face, facegeom, &fg);
    DMPlexPointLocalRead(dmCell, cells[0], cellgeom, &cgL);
    DMPlexPointLocalRead(dmCell, cells[1], cellgeom, &cgR);
    DMPlexPointLocalRead(dm, cells[0], x, &xL);
    DMPlexPointLocalRead(dm, cells[1], x, &xR);
    DMPlexPointGlobalRef(dm, cells[0], f, &fL);
    DMPlexPointGlobalRef(dm, cells[1], f, &fR);
    (*phys->riemann)(phys, fg->centroid, fg->normal, xL, xR, flux);
    for (i = 0; i < phys->dof; ++i) {
      if (fL) fL[i] -= flux[i] / cgL->volume;
      if (fR) fR[i] += flux[i] / cgR->volume;
    }
  }
}
```

Second Order TVD Finite Volume Method

Boundary Conditions

Boundary conditions are applied on marked faces

```
PetscErrorCode ApplyBC(DM dm, PetscReal time, Vec locX, User user) {
  VecGetArrayRead(user->facegeom, &facegeom);
  VecGetArray(locX, &x);
  for (fs = 0; fs < numFS; ++fs) {
    ModelBoundaryFind(mod,ids[fs],&bcFunc,&bcCtx);
    DMPlexGetStratumIS(dm, name, ids[fs], &faceIS);
    ISGetLocalSize(faceIS, &numFaces);
    ISGetIndices(faceIS, &faces);
    for (f = 0; f < numFaces; ++f) {
      const PetscInt face = faces[f], *cells;
      DMPlexPointLocalRead(dmFace, face, facegeom, &fg);
      DMPlexGetSupport(dm, face, &cells);
      DMPlexPointLocalRead(dm, cells[0], x, &xI);
      DMPlexPointLocalRef(dm, cells[1], x, &xG);
      (*bcFunc)(mod, time, fg->centroid, fg->normal, xI, xG, bcCtx);
    }
  }
}
```

Outline

The Great Solver Schism: Monolithic or Split?

Monolithic

- Direct solvers
- Coupled Schwarz
- Coupled Neumann-Neumann (need unassembled matrices)
- Coupled multigrid
- X Need to understand local spectral and compatibility properties of the coupled system

Split

- Physics-split Schwarz (based on relaxation)
- Physics-split Schur (based on factorization)
  - approximate commutators SIMPLE, PCD, LSC
  - segregated smoothers
  - Augmented Lagrangian
  - "parabolization" for stiff waves
- X Need to understand global coupling strengths

- Preferred data structures depend on which method is used.
- Interplay with geometric multigrid.

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);
```

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`
  - `-pc_fieldsplit_real_diagonal`
    Use diagonal blocks of operator to build PC
- Schur complements
  - `-pc_fieldsplit_schur_precondition <self,user,diag>`
    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)

```
-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)

```
-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)

```
-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)

```
-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

```
-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

```
-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

```
-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

```
-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_ksp_max_its 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

```
-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

```
-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

```
-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
```

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

Solver Configuration: No New Code

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

Additive Schwarz + Full Schur Complement

```
-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

FEM Setup

```
./bin/pythonscripts/PetscGenerateFEMQuadrature.py
  2 2 2 1 laplacian
  2 1 1 1 gradient
  src/snes/examples/tutorials/ex62.h
```

SNES ex62
Preconditioning

Jacobi

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 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
  -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
  -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 multiplicati
  -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
  -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
  -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
  -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
  -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

1. Sequential LU
   - ILUDT (SPARSEKIT2, Yousef Saad, U of MN)
   - EUCLID & PILUT (Hypre, David Hysom, LLNL)
   - ESSL (IBM)
   - SuperLU (Jim Demmel and Sherry Li, LBNL)
   - Matlab
   - UMFPACK (Tim Davis, U. of Florida)
   - LUSOL (MINOS, Michael Saunders, Stanford)
2. Parallel LU
   - MUMPS (Patrick Amestoy, IRIT)
   - SPOOLES (Cleve Ashcroft, Boeing)
   - SuperLU_Dist (Jim Demmel and Sherry Li, LBNL)
3. Parallel Cholesky
   - DSCPACK (Padma Raghavan, Penn. State)
   - MUMPS (Patrick Amestoy, Toulouse)
   - CHOLMOD (Tim Davis, Florida)
4. XYTlib - parallel direct solver (Paul Fischer and Henry Tufo, ANL)

3rd Party Preconditioners in PETSc

Complete table of solvers

1. Parallel ICC
   - BlockSolve95 (Mark Jones and Paul Plassman, ANL)
2. Parallel ILU
   - PaStiX (Faverge Mathieu, INRIA)
3. Parallel Sparse Approximate Inverse
   - Parasails (Hypre, Edmund Chow, LLNL)
   - SPAI 3.0 (Marcus Grote and Barnard, NYU)
4. Sequential Algebraic Multigrid
   - RAMG (John Ruge and Klaus Steuben, GMD)
   - SAMG (Klaus Steuben, GMD)
5. Parallel Algebraic Multigrid
   - Prometheus (Mark Adams, PPPL)
   - BoomerAMG (Hypre, LLNL)
   - ML (Trilinos, Ray Tuminaro and Jonathan Hu, SNL)

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**...

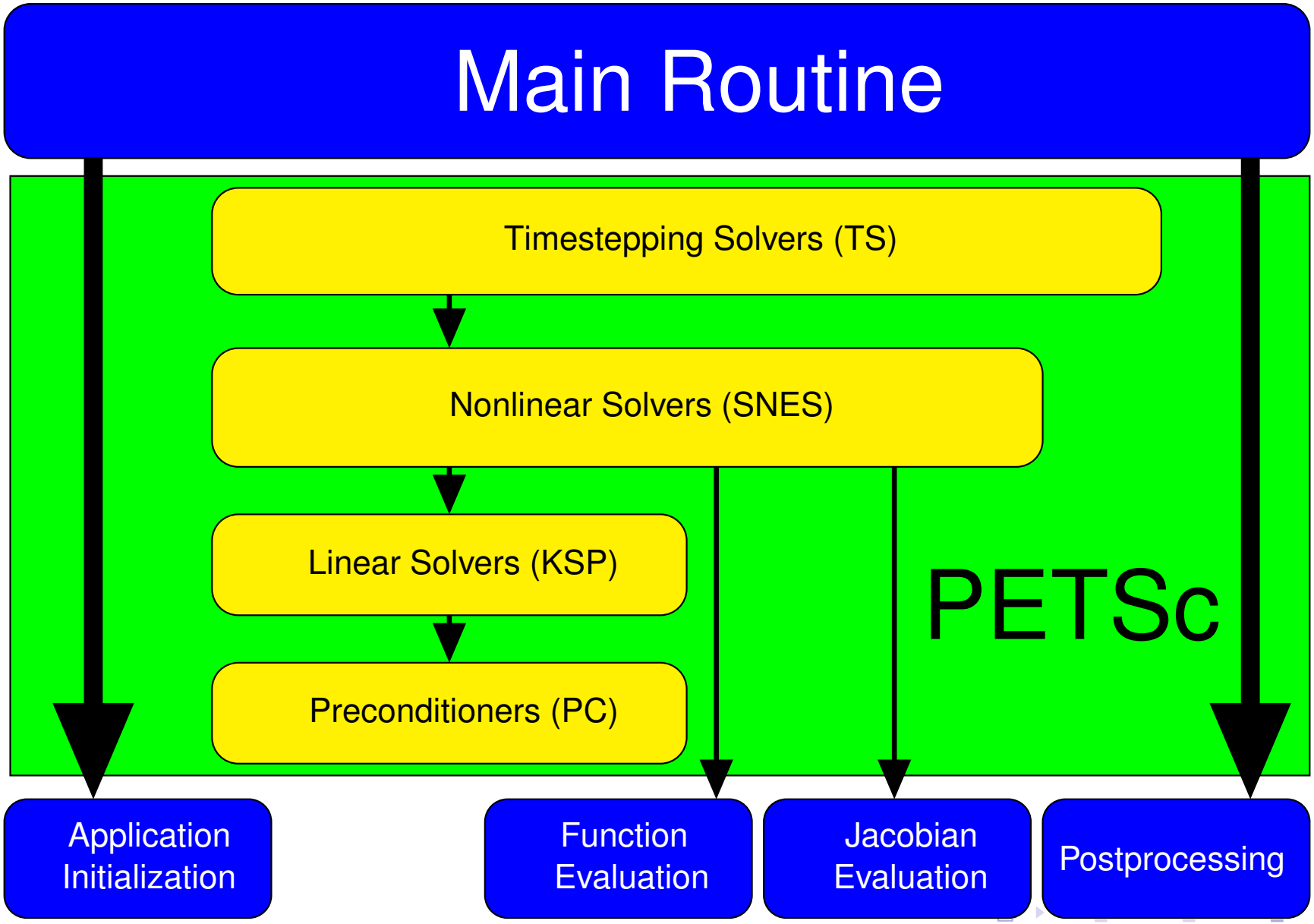
Always use **SNES**

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:

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

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

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
```

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 1e3
  -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 1e3
  -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_type test` and `-snes_mf_operator -pc_type lu`
- 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

Also called *globalization*

```
./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

Also called *globalization*

```
./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

Also called *globalization*

```
./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

Also called *globalization*

```
./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

Also called *globalization*

```
./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

Also called *globalization*

```
./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

Also called *globalization*

```
./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
```

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

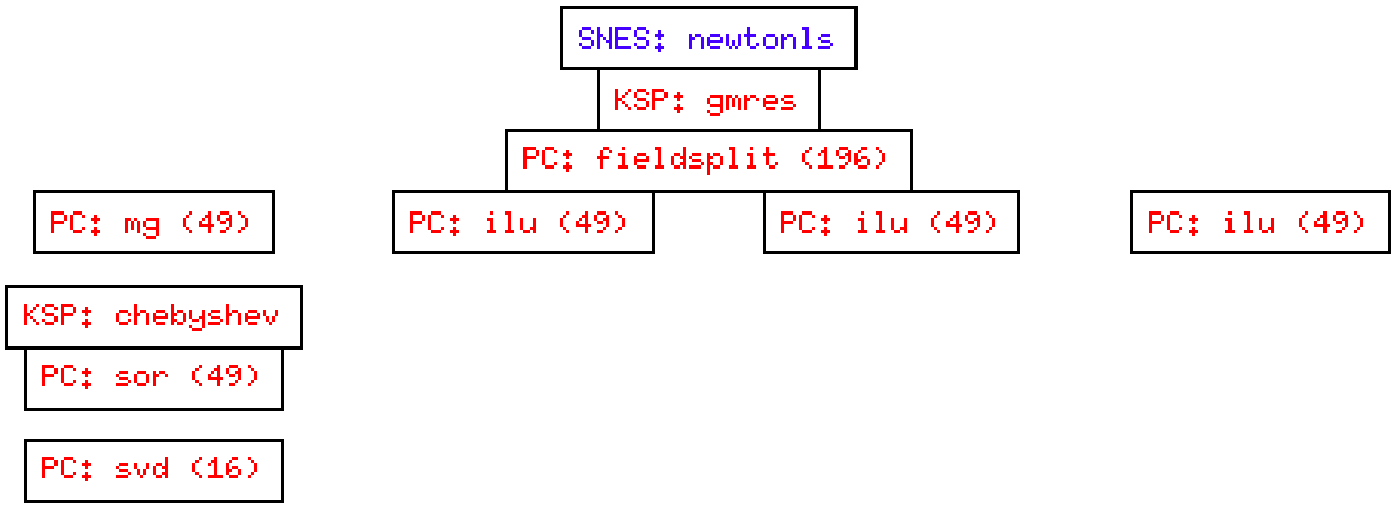
```
mpiexec -n 4 ./ex19 -da_refine 4 -snes_view
  -ksp_type fgmres -pc_type bjacobi -pc_bjacobi_blocks 2
  -sub_ksp_type gmres -sub_pc_type bjacobi -sub_ksp_max_it 2
  -sub_sub_ksp_type preonly -sub_sub_pc_type ilu

  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?

Didn't Time Integration Suck in PETSc?

Yes, it did . . .

until Jed, Emil, and Peter rewrote it $\Longrightarrow$

What about TS?

Didn't Time Integration Suck in PETSc?

Yes, it did . . .

until Jed, Emil, and Peter rewrote it $\Longrightarrow$

What about TS?

Didn't Time Integration Suck in PETSc?

Yes, it did . . .

until Jed, Emil, and Peter rewrote it $\Longrightarrow$

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