Using PETSc for scientific computing in parallel

Ed Bueler

Dept. of Mathematics and Statistics, UAF

19 June, 2007 (ARSC)

This talk comes out of joint work on PISM with
Jed Brown (now at ETH Switzerland),
Craig Lingle (GI UAF),
Dave Covey (GI UAF),
and Nathan Shemonski
(ARSC Summer 2007 REU Intern; Elon University NC)

Thanks to Don Bahls *at ARSC for help making it all work on the big machines.*

Outline

Outline

PETSc = Portable Extensible Toolkit for Scientific computation

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

PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

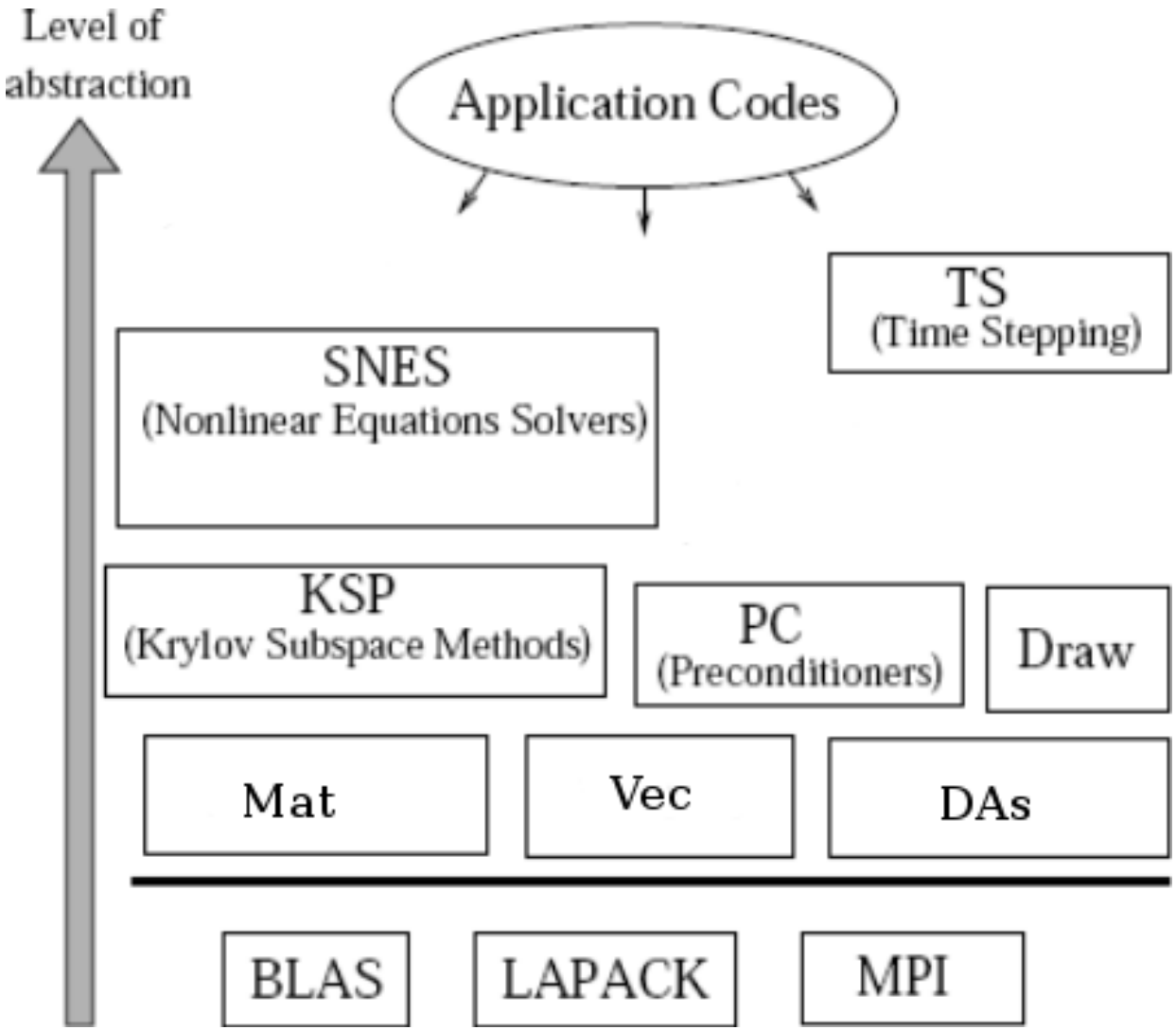
–Barry Smith, a PETSc developer since its start in 1991

PETSc

"PETSc Overview" section MOSTLY STOLEN FROM TUTORIAL SLIDES BY Matt Knepley:
http://www-unix.mcs.anl.gov/petsc/petsc-2/documentation/tutorials/PCFD2007.pdf

- download from `http://www.mcs.anl.gov/petsc` at Argonne National Lab (the MPI people)
- free for everyone, including industrial users
- hyperlinked manual, examples, and manual pages; hundreds of tutorial-style examples
- written in C but used from C, C++, Fortran 77/90, and Python; lots of Fortran users
- PETSc has run problems with over 500 million unknowns
- PETSc has run on over 6,000 processors efficiently
- PETSc applications have run at 2 Teraflops: LANL PFLOTRAN code

PETSc as a level of abstraction



PETSc uses MPI

- the Message Passing Interface is:
  - a library for parallel communication
  - a launcher for parallel jobs (`mpiexec`)
  - a community standard
- creators of MPI imagined most users would work at higher level of abstraction, *like the level in* PETSc
- you should *almost* never have to make MPI calls when using PETSc
- PETSc uses two default MPI communicators, yourself (`PETSC_COMM_SELF`) and everyone launched (`PETSC_COMM_WORLD`)
- PETSc uses MPI for point-to-point communication (like in `MatMult()`) and for reduction or scan operations among all processes (like in `VecDot()`)

extensions of PETSc

- libMesh, DEALII, Prometheus for meshing and finite element solns of PDEs
- TAO for optimization
- SLEPc for eigenvalue problems

configure and build PETSc: brief version

- set `PETSC_DIR` to the installation root directory
- run the configuration utility `$PETSC_DIR/config/configure.py`
  - there are many examples on the installation page
  - configuration files are placed in `$PETSC_DIR/bmake/$PETSC_ARCH`
  - `$PETSC_ARCH` has a default if not specified
- `make all test`
- try running PETSc examples first:
  - `cd $PETSC_DIR/src/snes/examples/tutorials`
  - `make ex5`
  - `./ex5`
  - `mpiexec -np 4 ./ex5`

Caveat about this talk

- PETSc is big. It is many things to many people.
- I will talk about the parts of PETSc I have used the most:
  - structured grids for finite differences and
  - linear algebra (as a part of the solution of nonlinear PDEs.)

Outline

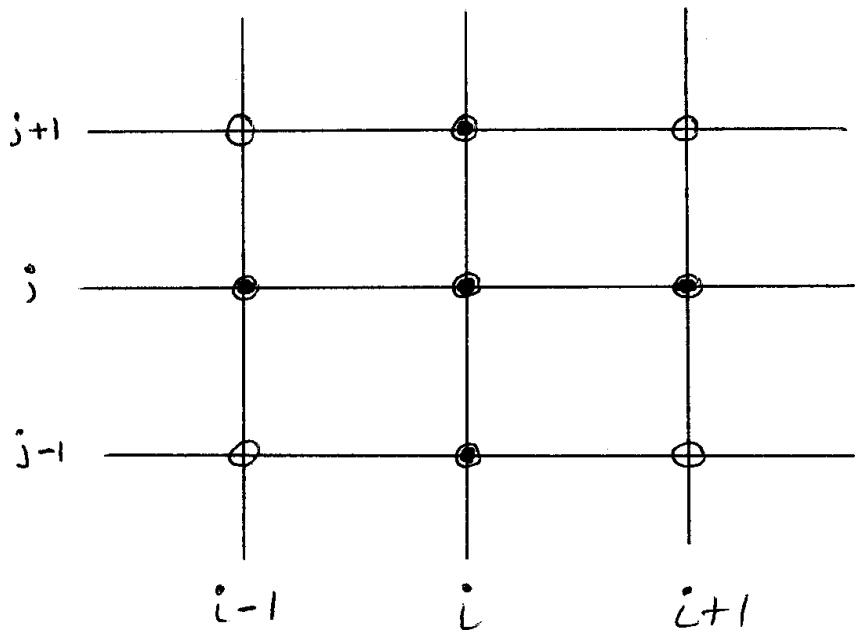
DA = Distributed Array

the DA abstract data type:

- is a structured grid interface
- is for fixed simple topologies
- supports stencils, communication, reordering
- is nice for simple finite differences

at right:

- circles are BOX stencil
- solid dots are STAR stencil

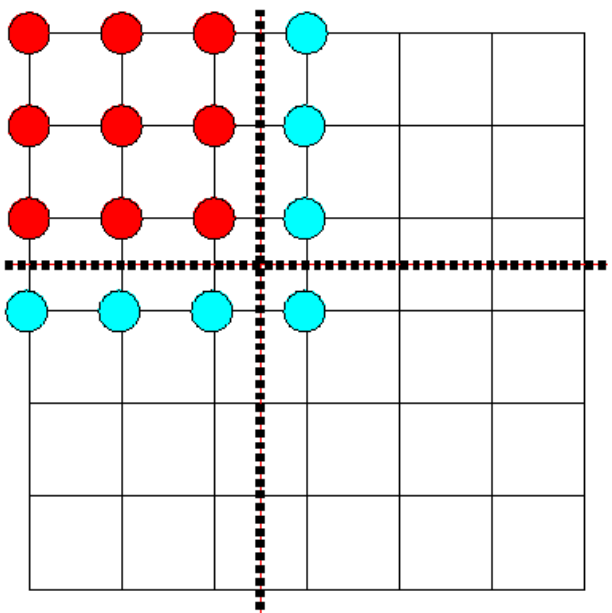


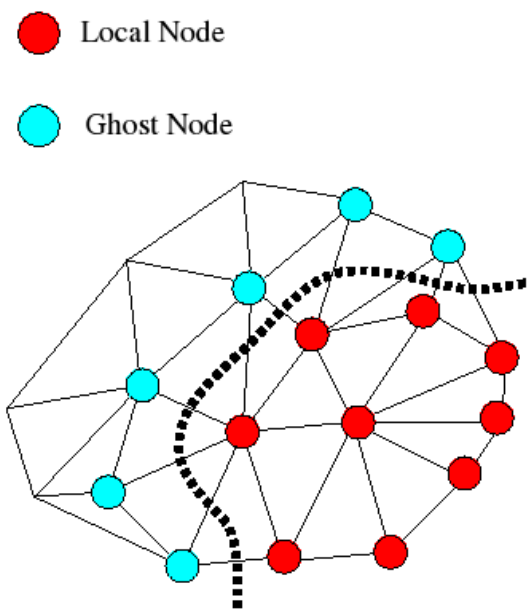
(there are also PETSc Mesh and DMMG abstract data types for finite elements and multigrid, resp.)

Ghosted points

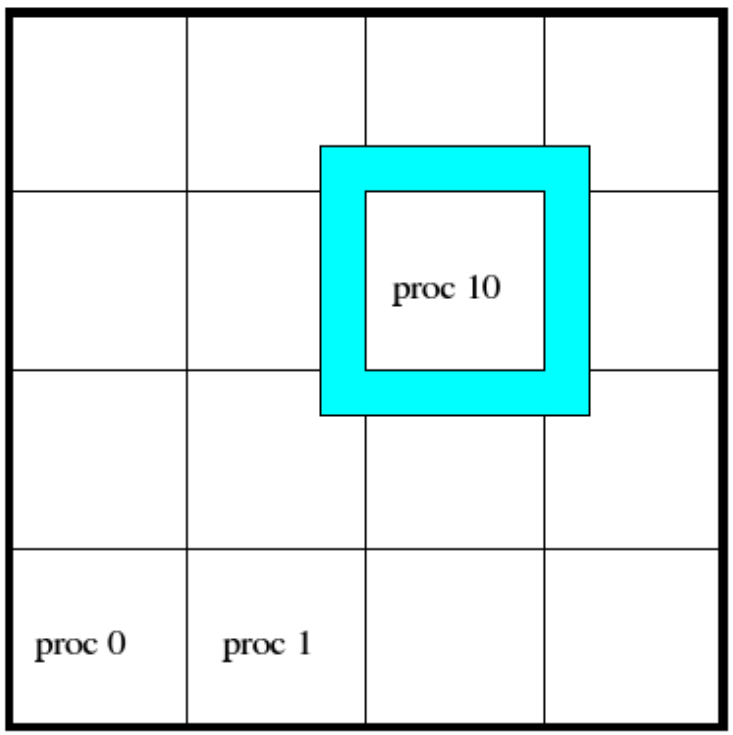
Why use DA abstraction?

- much parallel scientific computing is on a grid and has stereotyped communication
- one grid point needs to communicate with its neighbors at each time step, but not with distant grid points
- DAs manage this communication in *rectangular grid case* (left)
- no direct MPI calls to communicate values at neighbors on different processors

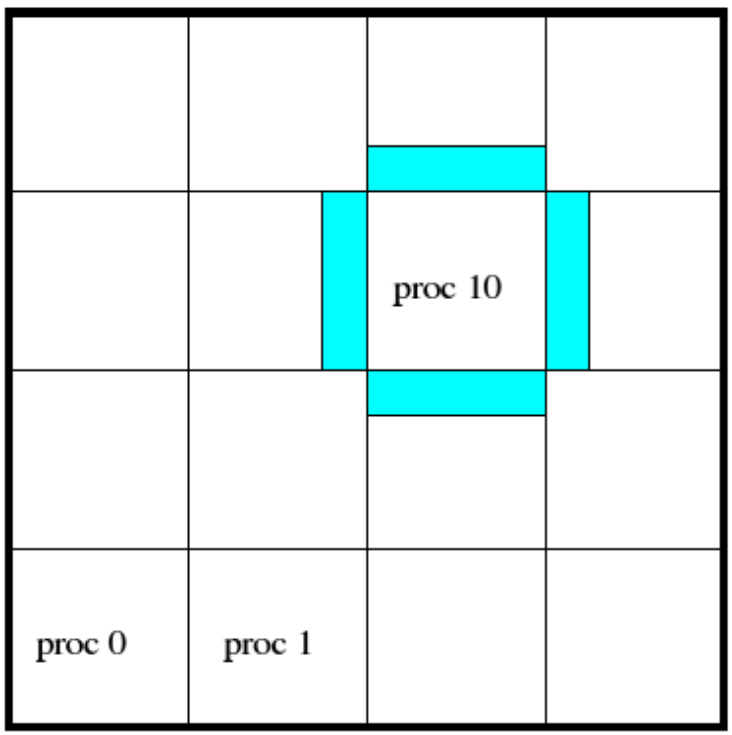




Inter-process communication for BOX versus STAR



Box Stencil



Star Stencil

Creating a 2d DA

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

wrap: Specifies periodicity
- DA_NONPERIODIC, DA_XPERIODIC, DA_YPERIODIC, or DA_XYPERIODIC

type: Specifies stencil
- DA_STENCIL_BOX or DA_STENCIL_STAR

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

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

dof: Degrees of freedom per node

s: The stencil width

lm/n: Alternative array of local sizes
- Use PETSC_NULL for the default

Creating a 3d DA

same, but with another dimension

DA manages a Vec

create DA, then creates Vecs managed by it; "local" Vecs carry their ghosted points while "global" Vecs do not; for example:

```
DA  da;
Vec v;

DACreate2d(com, DA_XYPERIODIC, DA_STENCIL_BOX,
    M, N, PETSC_DECIDE, PETSC_DECIDE,
    1, 1, PETSC_NULL, PETSC_NULL, &da);
DACreateLocalVector(da, &v);
for (k=0; k<steps; k++) { // e.g. time stepping loop
   [  MANIPULATIONS USING VALUES AT GHOSTED POINTS ]
   [  AND MODIFYING VALUES AT NON-GHOSTED POINTS        ]
   // now communicate and update ghosted points
   DALocalToLocalBegin(da, v, INSERT_VALUES, v);
   DALocalToLocalEnd(da, v, INSERT_VALUES, v);
}
VecDestroy(v);
DADestroy(da);
```

Outline

a program that does nothing, but in parallel!

nothin.c:

```
static char help[] = "A string for -help.\n";
#include <petscvec.h>
int main(int argc, char *argv[]) {
  PetscErrorCode  ierr;
  PetscMPIInt rank, size;

  ierr = PetscInitialize(&argc, &argv, PETSC_NULL, help); CHKERRQ(ierr);
  ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
  ierr = MPI_Comm_size(PETSC_COMM_WORLD, &size); CHKERRQ(ierr);

  ierr = PetscPrintf(PETSC_COMM_SELF,"rank = %d\n",rank);CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_WORLD,"size = %d\n",size);CHKERRQ(ierr);

  ierr = PetscFinalize(); CHKERRQ(ierr);
  return 0;
}
```

make a PETSc application

makefile:

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

nothin: nothin.o  chkopts
    -${CLINKER} -o nothin nothin.o ${PETSC_LIB}
```

run a PETSc application

```
$ make nothin
...
$ ./nothin
rank = 0
size = 1
$ mpiexec -n 4 ./nothin
rank = 0
size = 4
rank = 2
rank = 3
rank = 1
```

setting up a DA and a Vec

davec.c:

```
#include <petscda.h>

int main(int argc, char *argv[]) {
  PetscErrorCode  ierr;
  PetscMPIInt     rank;
  PetscInt        myM = 50, myN = 53;
  DA              da2;
  DALocalInfo     info;
  Vec             vv;

  PetscInitialize(&argc, &argv, PETSC_NULL, help);
  MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
  PetscOptionsGetInt(PETSC_NULL, "-M", &myM, PETSC_NULL);
  PetscOptionsGetInt(PETSC_NULL, "-N", &myN, PETSC_NULL);
```

setting up a DA and a Vec

davec.c cont.:

```
  DACreate2d(PETSC_COMM_WORLD, DA_XYPERIODIC, DA_STENCIL_STAR,
             myM, myN, PETSC_DECIDE, PETSC_DECIDE,
             1, 1, PETSC_NULL, PETSC_NULL, &da2);
  DAGetLocalInfo(da2, &info); CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_WORLD,
           "  whole grid size is %d x %d\n",myM,myN); CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_SELF,
           "  grid ranges for rank %d process:  xs=%d, xm=%d, ys=%d, ym=%d\n",
           rank,info.xs,info.xm,info.ys,info.ym);
  DACreateGlobalVector(da2, &vv);

  // could do something with vv here

  VecDestroy(vv);
  DADestroy(da2);
  PetscFinalize();
  return 0;
}
```

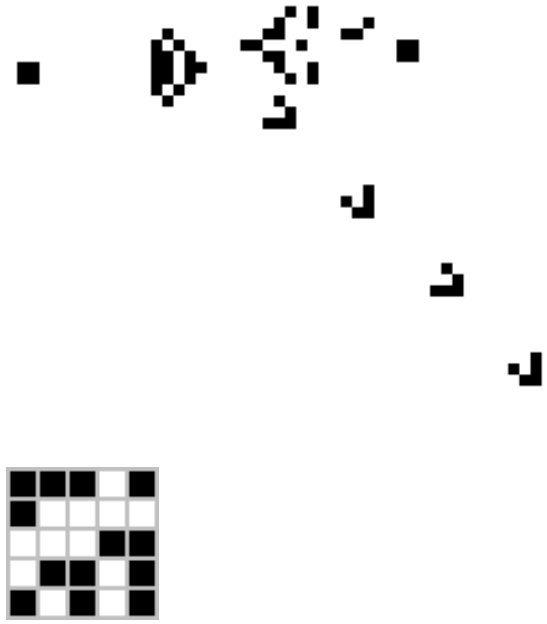
setting up a DA and a Vec

```
$ ./davec -M 200 -N 200
  whole grid size is 200 x 200
  grid ranges for rank 0 process:  xs=0, xm=200, ys=0, ym=200
$ mpiexec -n 6 ./davec -M 200 -N 200
  grid ranges for rank 1 process:  xs=100, xm=100, ys=0, ym=67
  grid ranges for rank 4 process:  xs=0, xm=100, ys=134, ym=66
  grid ranges for rank 5 process:  xs=100, xm=100, ys=134, ym=66
  whole grid size is 200 x 200
  grid ranges for rank 0 process:  xs=0, xm=100, ys=0, ym=67
  grid ranges for rank 2 process:  xs=0, xm=100, ys=67, ym=67
  grid ranges for rank 3 process:  xs=100, xm=100, ys=67, ym=67
```

John Conway's "Life" in parallel

Goal: Write a PETSc program using a `BOX` stencil `DA` for Life.

```
double lifeRule(double me, double nsum) {
   if (fabs(me - 0.0) < 0.1) { // dead cell
     if (fabs(nsum - 3.0) < 0.1)  return 1.0;
     else  return 0.0;
   } else { // live cell
     if (nsum < 1.9)  return 0.0;
     else if (nsum > 3.1)  return 0.0;
     else  return 1.0;
   }
}
```



"Life" in parallel

life.c:

```
int main(int argc, char *argv[]) {
  ...
  PetscScalar     startPattern[5][5] = {1.0, 1.0, 1.0, 0.0, 1.0,
                                       1.0, 0.0, 0.0, 0.0, 0.0,
                                       0.0, 0.0, 0.0, 1.0, 1.0,
                                       0.0, 1.0, 1.0, 0.0, 1.0,
                                       1.0, 0.0, 1.0, 0.0, 1.0};

  // start up and read options ...
  ierr = DACreate2d(PETSC_COMM_WORLD, DA_XYPERIODIC, DA_STENCIL_BOX,
            myM, myN, PETSC_DECIDE, PETSC_DECIDE, 1, 1, PETSC_NULL,
            PETSC_NULL, &da2); CHKERRQ(ierr);
  ierr = DAGetLocalInfo(da2, &info); CHKERRQ(ierr);
  ierr = DACreateGlobalVector(da2, &gstate); CHKERRQ(ierr);
  // set initial state ...
  // set ghosted values on each processor
  ierr = DACreateLocalVector(da2, &lstate); CHKERRQ(ierr);
  ierr = DAGlobalToLocalBegin(da2, gstate, INSERT_VALUES, lstate);
  ierr = DAGlobalToLocalEnd(da2, gstate, INSERT_VALUES, lstate);
  // view initial state ...
```

"Life" in parallel

life.c cont.:

```
  for (k = 0; k < max_steps; k++) {
    ierr = DAVecGetArray(da2, gstate, &st); CHKERRQ(ierr);
    ierr = DAVecGetArray(da2, lstate, &ost); CHKERRQ(ierr);
    for (i=info.xs; i<info.xs+info.xm; ++i) {
      for (j=info.ys; j<info.ys+info.ym; ++j) {
        nsum = ost[j][i+1] + ost[j-1][i+1] + ost[j-1][i] + ost[j-1][i-1] +
               ost[j][i-1] + ost[j+1][i-1] + ost[j+1][i] + ost[j+1][i+1]);
        st[j][i] = lifeRule(ost[j][i],nsum)
      }
    }
    ierr = DAVecRestoreArray(da2, lstate, &ost); CHKERRQ(ierr);
    ierr = DAVecRestoreArray(da2, gstate, &st); CHKERRQ(ierr);
    // update ghosted values from new (global) state on each processor
    ierr = DAGlobalToLocalBegin(da2, gstate, INSERT_VALUES, lstate);
    ierr = DAGlobalToLocalEnd(da2, gstate, INSERT_VALUES, lstate);
    ierr = VecView(gstate, viewer); CHKERRQ(ierr);
  }
  // finish up ...
}
```

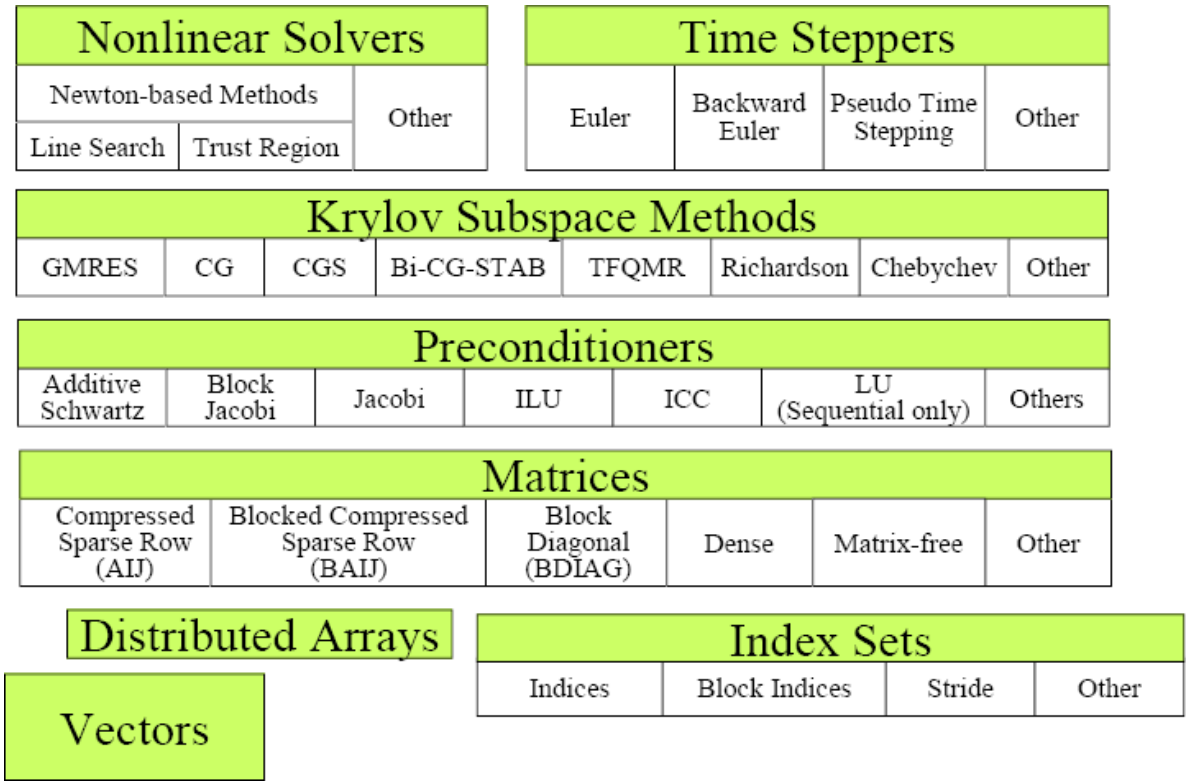
"Life" in parallel

```
$   make life

$   ./life

$   mpiexec -n 4 ./life -display :0
```

ED: RUN IT

Outline

Mathematical tools in PETSc



next in *this* talk

- matrices (`Mat` type)
- Krylov methods (`KSP` type)
- preconditioners

Linear algebra in parallel: Ideas

To solve $A\mathbf{x} = \mathbf{b}$:

- big problems frequently have sparse A
- so store A sparsely
- LU decomposition usually bad because fill-in
- Krylov subspace idea:
  - soln to $A\mathbf{x} = \mathbf{b}$ frequently well-approximated by element of subspace spanned by $\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^k\mathbf{b}\}$
  - so $\mathbf{x}$ well approximated by $p(A)\mathbf{b}$ where p is a polynomial
  - spectrum of A critical to ease of polynomial approximation
- precondition to get spectum in better position:
  $A\mathbf{x} = \mathbf{b} \quad \longrightarrow MA\mathbf{x} = M\mathbf{b}$
- PETSc *implements all these ideas* and *allows control of it at runtime*; examples next

Example of matrices and KSP

next few slides use `$PETSC_DIR/src/ksp/ksp/examples/tutorials/ex2.c`, an example code included in PETSc distribution, to illustrate `Mat` and `KSP` uses

- even on a regular grid, sometimes one needs to solve a set of equations
- values at distant grid points couple together through many equations

For example, finite difference approximation of Poisson eqn:

$$u_{xx} + u_{yy} = f$$

$$\downarrow$$

$$U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j} = (\text{known})$$

next: `ex2.c` sets up and solves this problem

Mat and KSP example

ex2.c:

```
#include "petscksp.h"

int main(int argc,char **args)
{
  Vec            x,b,u;  /* approx solution, RHS, exact solution */
  Mat            A;        /* linear system matrix */
  KSP            ksp;     /* linear solver context */
  PetscReal      norm;     /* norm of solution error */
  PetscInt       i,j,Ii,J,Istart,Iend,m = 8,n = 7,its;

  [ deleted: INITIALIZE AND GET OPTIONS ]
  MatCreate(PETSC_COMM_WORLD,&A);
  MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,m*n,m*n);
  MatSetType(A, MATAIJ);
  MatSetFromOptions(A);
  MatMPIAIJSetPreallocation(A,5,PETSC_NULL,5,PETSC_NULL);
  MatSeqAIJSetPreallocation(A,5,PETSC_NULL);
  MatGetOwnershipRange(A,&Istart,&Iend);
```

Mat and KSP example

ex2.c cont.:

```
for (Ii=Istart; Ii<Iend; Ii++) {
  v = -1.0; i = Ii/n; j = Ii - i*n;
  if (i>0)   {J = Ii - n; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES);}
  if (i<m-1) {J = Ii + n; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES);}
  if (j>0)   {J = Ii - 1; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES);}
  if (j<n-1) {J = Ii + 1; MatSetValues(A,1,&Ii,1,&J,&v,INSERT_VALUES);}
  v = 4.0; ierr = MatSetValues(A,1,&Ii,1,&Ii,&v,INSERT_VALUES);
}
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
VecCreate(PETSC_COMM_WORLD,&u);
VecSetSizes(u,PETSC_DECIDE,m*n);
VecSetFromOptions(u);
VecDuplicate(u,&b);
VecDuplicate(b,&x);
[ deleted: SET EXACT SOLUTION FOR COMPARISON ]
```

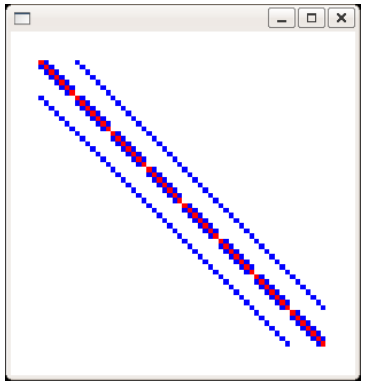
Mat and KSP example

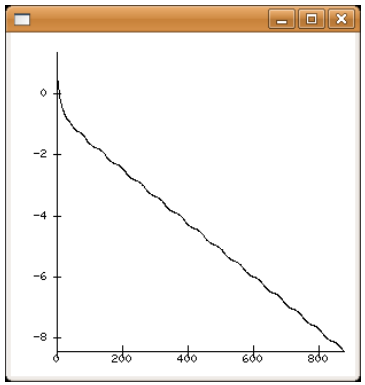
ex2.c cont. cont.:

```
  KSPCreate(PETSC_COMM_WORLD,&ksp);
  KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
  KSPSetTolerances(ksp,1.e-2/((m+1)*(n+1)),1.e-50,PETSC_DEFAULT,
                   PETSC_DEFAULT);
  KSPSetFromOptions(ksp);
  KSPSolve(ksp,b,x);
  [ deleted: COMPARE TO EXACT SOLUTION ]
  PetscPrintf(PETSC_COMM_WORLD,"Norm of error %A iterations %D\n",
              norm,its);
  KSPDestroy(ksp);
  VecDestroy(u);   VecDestroy(x);
  VecDestroy(b);   MatDestroy(A);
  PetscFinalize();
  return 0;
}
```

Controlling tolerance and method

PETSc allows control at command line of method and accuracy; also viewing of internal `Vec`, `Mat`, and `KSP` objects:






```
$  ./ex2
Norm of error 0.000156044 iterations 6
$  mpiexec -n 6 ./ex2
Norm of error 0.000709535 iterations 11
$ ./ex2 -ksp_type cgs
Norm of error 7.69507e-05 iterations 4
$ ./ex2 -ksp_rtol 1e-10
Norm of error 1.49481e-10 iterations 13
$ mpiexec -n 6 ./ex2 -m 200 -n 200
Norm of error 0.0051396 iterations 404
$ mpiexec -n 6 ./ex2 -m 200 -n 200 -pc_type asm
Norm of error 0.0041435 iterations 359
$ ./ex2 -mat_view_draw
Norm of error 0.000156044 iterations 6
$ ./ex2 -ksp_monitor_draw -ksp_rtol 1e-12 -m 300 -n 300
Norm of error 5.65271e-08 iterations 1127
```

Does result depend on number of processors?

Can a PETSc linear algebra result depend on number of processors?

- *yes*, sometimes
- the KSP act the same on different numbers of processors
- the *preconditioner* acts differently
- one may *turn off the preconditioner* to get independence of number of processors; generally slower convergence without preconditioner

```
$  mpiexec -n 1 ./ex2
Norm of error 0.000156044 iterations 6
$  mpiexec -n 6 ./ex2
Norm of error 0.000709535 iterations 11
$  mpiexec -n 1 ./ex2 -pc_type none
Norm of error 1.68964e-05 iterations 13
$  mpiexec -n 6 ./ex2 -pc_type none
Norm of error 1.68964e-05 iterations 13
```

No conclusion. Just back to work . . .

- talk to me about PETSc anytime, or email the maintainers, *who do check their email*!
- codes `nothin.c`, `davec.c`, `life.c`, `ex2.c` all available [HOW?]
- many problems don't need PETSc, as *Matlab* will do them just fine
- numerical algorithm improvements usually more effective than parallelization

Thanks for your attention or tolerance!