

# Practical PETSc Tutorial

Patrick Sanan

[patrick.sanan@usi.ch](mailto:patrick.sanan@usi.ch), [erdw.ethz.ch](mailto:patrick.sanan@erdw.ethz.ch)

USI Lugano / ETH Zürich

November 30, 2016



**ETH** zürich



# This Tutorial

- ▶ We will use PETSc as an extended example of working with a library
  - ▶ This is because I am familiar with it, and because its flexible design allows us to examine many things quickly
  - ▶ All libraries present limitations and annoyances, and no library is a silver bullet: PETSc is no exception
- ▶ We don't have time for a full tutorial on the library, unfortunately
- ▶ I have adapted some “miniapp” code to use PETSc so that you can see familiar concepts in a new framework
- ▶ The majority of the time will be spent experimenting with this code: libraries allow you to quickly leverage functionality that would be time-consuming to “roll yourself”, and PETSc is an extreme example of this in that you can even do this experimentation at runtime
- ▶ Please ask questions at any time.

# What is PETSc?

- ▶ Origins as one of the first success stories of MPI, a library for domain decomposition-based PDE solvers
- ▶ Extended to provide a full set of tools for solving large-scale discretized PDE in distributed-memory parallel environments
- ▶ Distributed linear algebra, linear solvers, preconditioners, nonlinear solvers, timesteppers, domain management tools, optimization tools (TAO), and associated utilities.
- ▶ Over 20 years of development, fully supported, based at Argonne National Lab
- ▶ Written in object-oriented C, with a Fortran interface <sup>1</sup>
- ▶ Forms the basis for many of the libraries we've seen in the previous lecture, especially higher-level PDE libraries

---

<sup>1</sup>and see `petsc4py`, which offers a Python interface

# Why Use PETSc?

- ▶ Write robust, scalable MPI codes to solve PDE, without writing much MPI code yourself
- ▶ Use a combinatorial explosion of solvers, configurable at runtime
- ▶ Run your code essentially anywhere, from your laptop to Piz Daint
- ▶ Configure with a huge number of external packages (including external linear solvers)
- ▶ Excellent support and community
- ▶ Open source, free software (2-clause BSD license)

# What's in a Name?

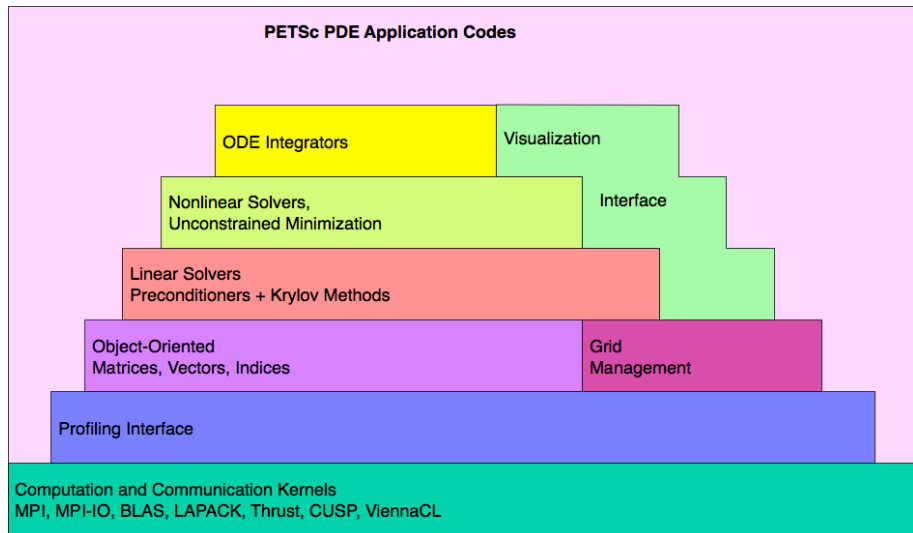
- ▶ **P**ortable
- ▶ **E**xtensible
- ▶ **T**oolkit for
- ▶ **S**cientific
- ▶ **c**omputation

An alternate acronym: the “**P**ortable, **E**xtensible **T**oolkit for **S**olver composition”

# Where to Learn More

- ▶ The PETSc website  
[mcs.anl.gov/petsc](http://mcs.anl.gov/petsc)
- ▶ The manual  
[mcs.anl.gov/petsc/documentation/index.html](http://mcs.anl.gov/petsc/documentation/index.html)
- ▶ Some simple hands-on tutorials  
[mcs.anl.gov/petsc/documentation/tutorials/HandsOnExercise.html](http://mcs.anl.gov/petsc/documentation/tutorials/HandsOnExercise.html)
- ▶ The mailing lists  
[mcs.anl.gov/petsc/miscellaneous/mailling-lists.html](http://mcs.anl.gov/petsc/miscellaneous/mailling-lists.html)
- ▶ The man pages, and linked examples  
[mcs.anl.gov/petsc/documentation/index.html](http://mcs.anl.gov/petsc/documentation/index.html)

# PETSc Components



# ICS Exercise 1: Hello World

Log on to the ICS Cluster and obtain the following example

```
git clone https://bitbucket.org/psanan/petsc_test_ics
```

Run it

1. On the login node
2. With a batch script on the compute nodes
3. In an interactive session



- ▶ We will use a PDE used for the CSCS Summer School <sup>2</sup>, Fischer's Equation in 2 dimensions

$$\frac{\partial s}{\partial t} = D \left( \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} \right) + Rs(1 - s)$$

discretized the same way on the same domain, a rectangle equipped with a regular grid, using standard finite differences / finite volumes.

- ▶ We will solve the same system, but using PETSc's abstractions
- ▶ We will leverage a powerful design component of PETSc, namely that it is **runtime configurable**, meaning that we can select from a huge number of solvers at runtime and quickly see how they perform
- ▶ We will write straightforward code which nevertheless will scale to large numbers of MPI processes (without us writing much MPI)

---

<sup>2</sup><https://github.com/eth-cscs/SummerSchool2016>

# Porting to PETSc

As a short example, code which applies a finite difference Laplacian to a vector (similar, but not identical, to what we do to port the miniapp)

```
#include <petscdmda.h>
#include "ctx.h"

#ifdef __FUNCT__
#define __FUNCT__ "applyA"
PetscErrorCode applyA(Mat A, Vec in, Vec out)
{
    PetscErrorCode    ierr;
    Ctx                *ctx;
    const PetscScalar **inarr;
    PetscScalar        val,**outarr;
    PetscInt           i,j,ixs,iys,ixm,iym,imin,imax,jmin,jmax,M,N;
    PetscBool          left,right,up,down;
    Vec                in_local;
    PetscReal          oneoverhy2,oneoverhx2;

    PetscFunctionBeginUser;

    ierr = MatShellGetContext(A, &ctx);CHKERRQ(ierr);
    M      = ctx->M;
    N      = ctx->N;
    in_local = ctx->work_local[0];
    oneoverhy2 = 1.0/(ctx->hy*ctx->hy);
    oneoverhx2 = 1.0/(ctx->hx*ctx->hx);

    /* Scatter global-->local to have access to the required ghost values */
    ierr=DMGlobalToLocalBegin(ctx->da,in,INSERT_VALUES,in_local);CHKERRQ(ierr);
    ierr=DMGlobalToLocalEnd  (ctx->da,in,INSERT_VALUES,in_local);CHKERRQ(ierr);
}
```

# Porting to PETSc (continued)

```
/* Get the boundaries of the local subdomain */
DMDAGetCorners(ctx->da, &ixs, &iys, 0, &ixm, &iym, 0);CHKERRQ(ierr);

/* Get access to the raw arrays (with ghosts).
   Note that PETSc allows these to be accessed with *global* indices */
DMDAVecGetArray(ctx->da,out,&outarr);CHKERRQ(ierr);
DMDAVecGetArrayRead(ctx->da,in_local,&inarr);CHKERRQ(ierr);

/* Determine active (global) boundaries */
up   = (iys == 0); jmin = up ? iys + 1 : iys;
down = (iys + iym == N); jmax = down ? iys + iym - 1 : iys + iym;
left  = (ixs == 0); imin = left ? ixs + 1 : ixs;
right = (ixs + ixm == M); imax = right ? ixs + ixm - 1 : ixs + ixm;

/* Handle corners */
if(up && left){
    val=0;
    j=0; i=0;
    val+=inarr[j][i+1] * (-oneoverhx2);
    val+=inarr[j+1][i] * (-oneoverhy2);
    val+=inarr[j][i] * 2.0 * (oneoverhy2 + oneoverhx2);
    outarr[j][i]=val;
}
if(up && right){
    /* ... */
}
if(down && left){
    /* ... */
}
if(down && right){
    /* ... */
}
```

# Porting to PETSc (continued)

```
/* Handle edges (excluding corners ) */
if (up){
    j=0;
    for (i=imin; i<imax; ++i) {
        val=0;
        val+=inarr[j ][i-1] *      (-oneoverhx2);
        val+=inarr[j ][i+1] *      (-oneoverhx2);
        val+=inarr[j+1][i ] *      (-oneoverhy2);
        val+=inarr[j ][i ] * 2.0 * (oneoverhy2 + oneoverhx2);
        outarr[j][i]=val;
    }
}

if (down){
    /* ... */
}

if (left){
    /* ... */
}

if (right){
    /* ... */
}
```

# Porting to PETSc (continued)

```
/* Handle the interior points */
for (j=jmin; j<jmax; ++j) {
    for (i=imin; i<imax; ++i) {
        val=0;
        val+=inarr[j-1][i] *      (-oneoverhy2);
        val+=inarr[j][i-1] *      (-oneoverhx2);
        val+=inarr[j][i+1] *      (-oneoverhx2);
        val+=inarr[j+1][i] *      (-oneoverhy2);
        val+=inarr[j][i] * 2.0 * (oneoverhy2 + oneoverhx2);
        outarr[j][i]=val;
    }
}

/* Revoke access to raw arrays */
DMDAVecRestoreArray(ctx->da,out,&outarr);CHKERRQ(ierr);
DMDAVecRestoreArrayRead(ctx->da,in_local,&inarr);CHKERRQ(ierr);

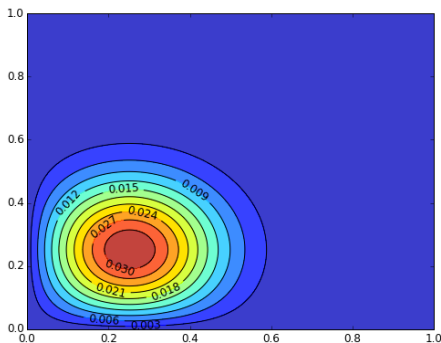
PetscFunctionReturn(0);
}
```

## Exercise 1: Run the Code

- ▶ Obtain the miniapp code from the course directory or with

```
git clone https://  
    bitbucket.org/  
    psanan/  
    ics_petsc_miniapp
```

- ▶ Follow the instructions in README.md
- ▶ You should be able to produce an image like the one on the right



Aside: we are using the `petsc` module (thanks, Radim!), which frees us from having to configure and compile PETSc. If you would like to use additional external packages, you can configure and build PETSc yourself.

- ▶ PETSc allows a “top-down” approach, so we begin by discussing how to adapt the miniapp to use PETSc ODE/DAE solver object, called TS
- ▶ The advantages of doing things this way include
  - ▶ The usual advantage of top-down design: having a clear view of the task, not implementing details until they are needed, etc.
  - ▶ Additional flexibility, as you can use PETSc’s objects for more of the code
- ▶ Let’s take a look at `main.c`

## Exercise 2: Changing the ODE solver

- ▶ Run `./main -help` to see (many) possible command line options
- ▶ Experiment running the program
  - ▶ With different numbers of grid points
  - ▶ With different numbers of time steps
  - ▶ With different numbers of MPI processes
- ▶ Try the `-ts_view` and `-ts_monitor` options
- ▶ Try using different timesteppers, using `-ts_type`.
  - ▶ See the list of types in the [manual](http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/TS/TSType.html) or on the man page for TSType at <http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/TS/TSType.html>
  - ▶ Not all will work immediately (some require you to provide more information about your system), but experiment with some standard choices like `rk`, `theta`, `ssp`, `bdf`, etc.



# Code Walkthrough : Distributed Vectors, Array, and Linear Operators

- ▶ See `system.c`
- ▶ We will see how to assemble a Jacobian matrix
- ▶ PETSc also supports custom “matrix-free” operators <sup>3</sup>, and can estimate the Jacobian for you using finite differences.

---

<sup>3</sup>See [MatShell](#)

- ▶ PETSc objects can be “viewed” in various ways
- ▶ This includes writing to the screen, to files, or even to network sockets
- ▶ See `dump.c`

## Exercise 3: Parallel Preconditioners

- ▶ Use the `-assemble` option
- ▶ Using `-ts_view`, determine what the default preconditioner (PC) is for the linear solver (KSP)
- ▶ Use `-ksp_monitor` and describe what happens to the convergence as you strong scale (increase the number of MPI ranks for the same problem size)
- ▶ Experiment with another preconditioner, an additive Schwarz method, with `-pc_type asm`. Note that adding `-help` will now give you more options related to this preconditioner <sup>4</sup>

---

<sup>4</sup>The `-help` output can get long: try `./main -help -pc_type asm | grep asm`

## Exercise 4: Bigger Time Steps

- ▶ Experiment with command line options to increase the time step to, say, 1
- ▶ Try to reduce the number of linear solver iterations by using a strong preconditioner like `-pc_type gamg`<sup>5</sup>

---

<sup>5</sup>This is algebraic multigrid

- ▶ `-log_summary`<sup>6</sup> provides a wealth of information, and is a necessary companion to allow quick interpretation of experiments with different solvers
- ▶ Includes
  - ▶ Time and flops
  - ▶ Call counts
  - ▶ Load balances
  - ▶ Cumulative memory usage (**Not** high-water mark)

---

<sup>6</sup>called `-log_view` in more recent versions of PETSc

## Exercise 5: Algorithmic Experimentation

- ▶ Using only the command line options and `-log_summary`, see how much you can speed up the code
- ▶ Examine strong-scaling behavior (how does increasing the number of processes affect the solution time?)

## Exercise 6 (Bonus): Writing a matrix-free operator application

- ▶ Look at the documentation for `MatShell`, `MatShellSetOperation()`, etc. (See the man pages, linked examples, and Section 3.3 of the manual)
- ▶ Using the example from earlier in the slides as an example, write a matrix-free operator which applies the Jacobian in the miniapp, and confirm that it gives the same results as using the assembled operator.

# Continuing with PETSc

- ▶ See the manual and other documentation at [www.mcs.anl.gov/petsc/documentation/index.html](http://www.mcs.anl.gov/petsc/documentation/index.html)
- ▶ Learn how to use the mailing lists! See [www.mcs.anl.gov/petsc/miscellaneous/mailling-lists.html](http://www.mcs.anl.gov/petsc/miscellaneous/mailling-lists.html).
  - ▶ Questions are answered quickly here
  - ▶ General questions : `petsc-users`
  - ▶ Developer topics (if planning to contribute) : `petsc-dev`
  - ▶ Private queries, bugs, installation problems : `petsc-maint`
  - ▶ For best results:
    1. Include the entire error message
    2. Include `configure.log` and `make.log` if sending a configure/install problem to `petsc-maint`
    3. The more specific, the better. If you can provide code to reproduce your problem, that is best.
    4. Note the archives
- ▶ Note the large number of examples (see links from the man pages)
- ▶ Up-and-coming resource: [scicomp.stackexchange.com](http://scicomp.stackexchange.com)



## ICS Exercise 2: Reconstructing an Example

The aim is to translate the comments in a PETSc example back into PETSc code.

```
git clone https://bitbucket.org/psanan/  
ics_petsc_ex23_fillin.git
```

- ▶ Fill in the `/* TODO */`'s in the code, using the PETSc references mentioned above, and the help of the instructor.
- ▶ Run the code in parallel on the ICS cluster
- ▶ Try changing the preconditioner
- ▶ Run with `-log_view` and see how your preconditioner affects the timing

(When finished, you should end up with something that resembles KSP tutorial example 23 from PETSc, at <http://www.mcs.anl.gov/petsc/petsc-3.5/src/ksp/ksp/examples/tutorials/ex23.c.html>.)