Università della Svizzera italiana

Faculty of Informatics

Institute of Computational Science ICS

Platform for Advanced Scientific Computing

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Practical PETSc Tutorial

Patrick Sanan
Advanced Scientific Computing Group, USI Lugano

sananp@usi.ch

Why use PETSc?

- Write robust, scalable PDE codes without the pain of writing MPI.

- Use a combinatorial explosion of solvers, configurable at runtime.

- Run your code anywhere, from your laptop to Piz Daint

- Configure with a huge number of external packages (including external linear solvers)

- Excellent support and community (but documentation that could still use some polishing)

- **P**ortable : compile-anywhere C (and Fortran) with a robust configuration process
- **E**xtensible : plug-in oriented design, so you can write your own implementations of any object, from a vector to an ODE solve
- **T**oolkit for **S**cientific **c**omputation : originally developed on top of MPI to allow for domain-decomposition approaches to solving numerical PDE based on sparse matrix operations, now provides that and more!

See the PETSc web site for tutorials, the manual, etc.
mcs.anl.gov/petsc

Here we will quickly move to some example code!

PETSc Mini-app

- We will use the same simple nonlinear PDE (Fischer's equation) that you have become familiar with during the week.

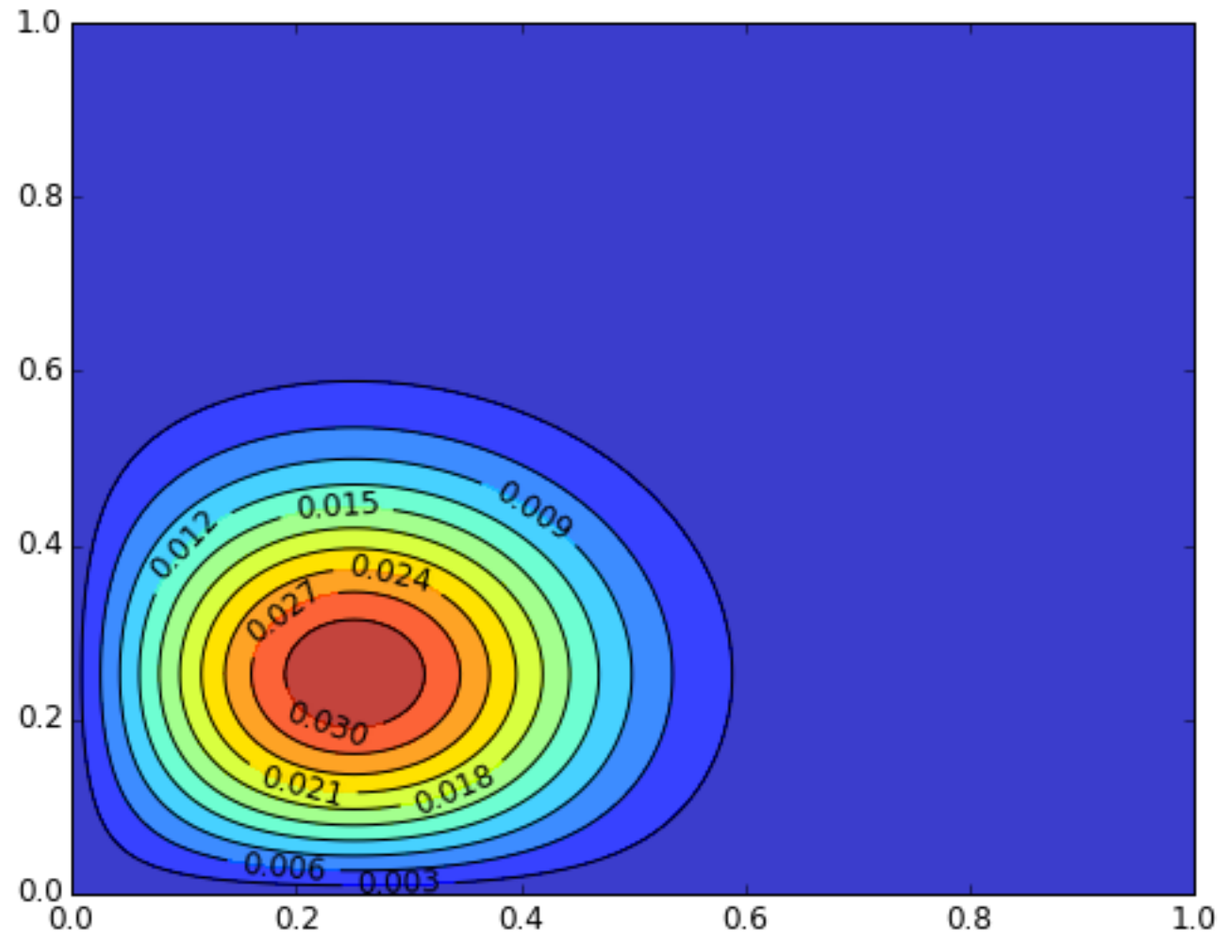
$$\frac{\partial s}{\partial t} = D\overbrace{\left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2}\right)}^{\text{diffusion}} + \overbrace{Rs(1-s)}^{\text{reaction/growth}}$$

- A key design point of PETSc is that it is run-time configurable.
- This means that we can completely change our solvers **from the command line**.
- We can easily change the linear solver, add preconditioners, change the nonlinear solver, or change the timestepping scheme.
- Further, we write a straightforward code that can **scale to huge numbers of MPI processes** with a uniform interface which hides the details of the halo exchanges and other communication details.

Exercise 1

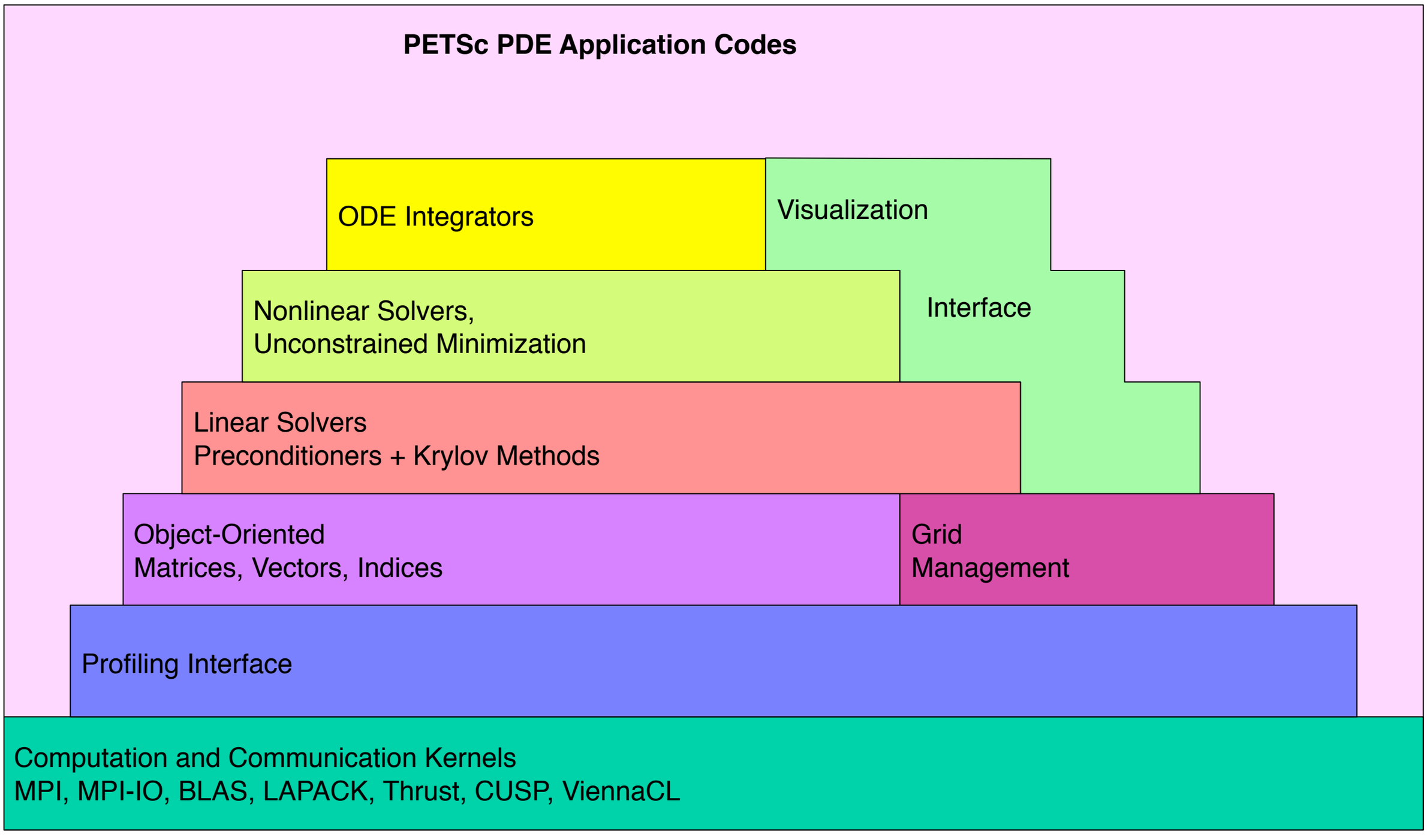
Get the code running!

- Obtain the source
- Follow the instructions in `libraries/petsc-miniapp/README.md`



(Note that it is very convenient to use the **`cray-petsc`** module, as we will do here, but that you can, with relative ease, build a more recent version of the library on essentially any machine, and use/modify the included makefile.local)

We will touch (briefly) on all the major components pictured here



Code Walkthrough: ODE Solvers

see `main.c`

(Note that PETSc's TS also includes DAE solvers, though we only consider ODE here)

Exercise 2

Change the ODE integrator

- Experiment running the program with different numbers of grid points, time steps, and MPI processes
- Experiment with the command line options
  `-ts_view`, **`-ts_monitor`**
- Experiment with **`-ts_type`** (Not all will work - later on as a bonus exercise we can change this)

Table 10: Time integration schemes.

TS Name	Reference	Class	Type	Order
euler	forward Euler	one-step	explicit	1
ssp	multistage SSP [20]	Runge-Kutta	explicit	≤ 4
rk*	multistage	Runge-Kutta	explicit	≥ 1
beuler	backward Euler	one-step	implicit	1
cn	Crank-Nicolson	one-step	implicit	2
theta*	theta-method	one-step	implicit	≤ 2
alpha	alpha-method [18]	one-step	implicit	2
gl	general linear [5]	multistep-multistage	implicit	≤ 3
eimex	extrapolated IMEX [7]	one-step	≥ 1, adaptive	
arkimex	see Tab. 12	IMEX Runge-Kutta	IMEX	$1-5$
rosw	see Tab. 13	Rosenbrock-W	linearly implicit	$1-4$

http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf

Code Walkthrough: Distributed Vectors, Arrays, and Linear Operators

see **`system.c`**

(We see how to assemble an operator, but PETSc also supports matrix-free operators, as in the miniapp, with the **`MatShell`** object)

Code Walkthrough: Simple Viewing

see **dump.c**

Exercise 3

Parallel Preconditioners

- Use the **`-assemble 1`** option
- 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)
  - Hint: perform one time step and use **`-ksp_view`**, and look at the preconditioner being used
- Experiment with an additive Schwarz preconditioner with options like
  `-help –pc_type asm`

and learn how to set the subsolver type (note that **`-help`** should now give you options specific to the ASM preconditioner)

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
  - Try `-pc_type gamg`

(Challenging) **Bonus exercise for the bored or bold**
PETSc includes many very sophisticated “Implicit Explicit” integrators. Take a look in the PETSc manual and see if you can adapt the example to use TSARKIMEX.

Performance Profiling

-log_summary provides a wealth of information.

- Time and flops used by various functions
- Call counts
- load balances
- Memory usage
- Much more!

Exercise 5

Algorithmic experimentation

- See how much you can speed up the execution of the program just by selecting different options at the command line
- For a reasonable set of options, vary the number of MPI processes and use **`-log_summary`** to examine the strong-scaling efficiency (does doubling the number of process halve the time?)

Ask questions!

Also, note the manual and man pages at http://www.mcs.anl.gov/petsc/documentation/index.html

Continuing with PETSc

- Read the manual
- join the petsc-users mailing list (and petsc-dev if interested)
  - http://www.mcs.anl.gov/petsc/miscellaneous/mailing-lists.html
  - Note the archives
- Look at the many examples in the **`src/`** tree
- Ask for help on the petsc-users list, or at petsc-maint@mcs.anl.gov (if you must)
- To get the best help
  - Read the error message
  - In emails, include full error messages, **`configure.log`** and **`make.log`**, and the output of **`-log_summary`** if asking about performance.
- Have fun, and let me know how it goes, as I am working on a longer PETSc tutorial ! (sananp@usi.ch)