PRACE PETSc Tutorial

CSC - IT Center for Science, Espoo, Finland

2-3 May, 2019

Part 1: Introduction & Hello World

Václav Hapla
ETH Zürich

Outline of the tutorial

Day 1 – Linear algebra and linear system solution

1. **Introduction & Hello World**
2. **Objects**
3. **Vectors & Index Sets**
4. **Matrices**
5. **Linear System Solvers**

Day 2 – Advanced topics

6. **Technical Stuff** (installation from source, use as package manager, start your own project, debug, profile)
7. **Discretization** (DM)
8. **Advanced Solvers** (SNES, TAO, SLEPc)
9. **Extras** (MATLAB interface, individual projects)

Few practical notes

- Questions welcome!
- Ask any time.
- If you are too quick during hands-ons
  - figure out your own additional tasks
  - try to find corresponding routines in PETSc documentation
  - maybe something you need in your practice

Frameworks for numerical computing

- „A **software framework** is a software providing **generic functionality** that can be selectively changed by user code, thus providing **application specific software.**“ (wikipedia.org)
- There are only few such ones for **numerical computing**, e.g. **Trilinos** and **PETSc**.
- **Don‘t reinvent the wheel!**
- many **numerical algorithms** (LU, CG, SPMV, …) **implemented** and **tested**
- focus on **novel** algorithms with an **added value**

What is PETSc

- **building blocks** (data structures and routines) for the scalable **parallel** solution of **scientific applications,** mainly **PDE-based**
- allows thinking in terms of **high-level objects** (matrices) instead of low-level objects (raw arrays)
- coded primarily in **C language** but good **FORTRAN** support, can also be called from **C++**, **Python** and **Java** codes
- highly **portable**
- source code and mailing lists **open to everybody**
- homepage: www.mcs.anl.gov/petsc

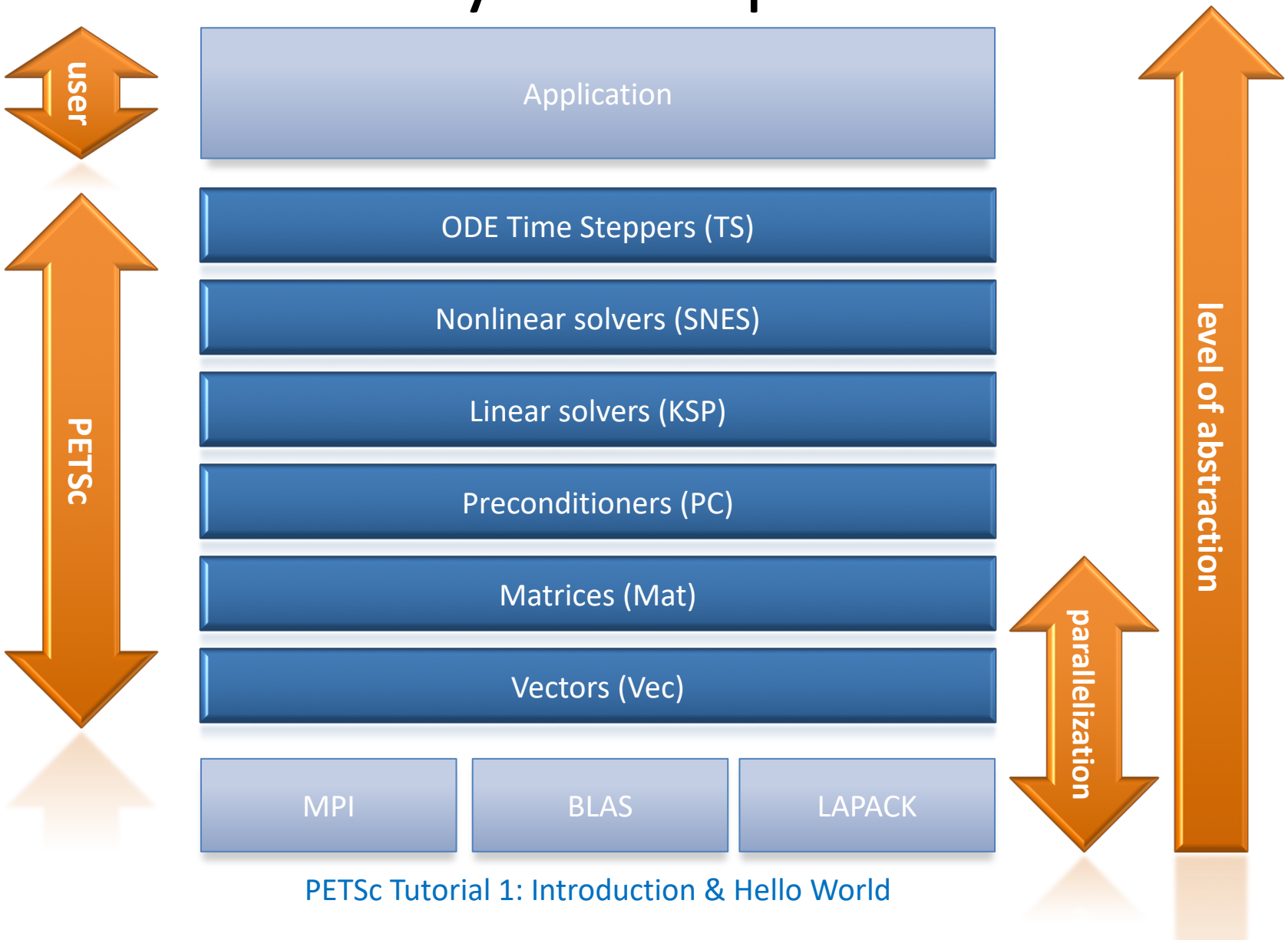
PETSc mission

„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

(PETSc founder)

Hierarchy of components



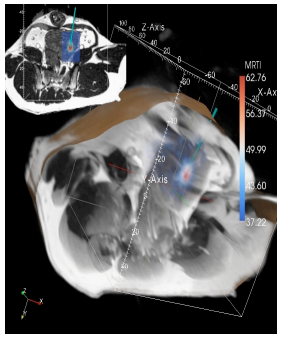
Parallelism in PETSc

- PETSc is parallelized mostly using **MPI**
- **MPI** provides **low-level routines** to **exchange data primitives** between processes
- **PETSc** provides **mid-level routines** such as
  - insert matrix element to arbitrary location
  - parallel matrix-vector product
- you can **call MPI** directly if needed
- **same code** for **sequential** and **parallel** runs
- support for hybrid **MPI + {shared memory, accelerator} parallelism**
  - **thread-safe** but not **threaded**
  - **GPU-accelerated matrix types** `MATAIJCUSPARSE, MATAIJVIENNACL`

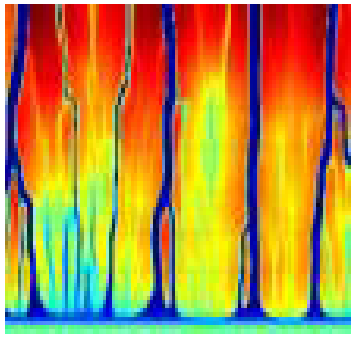
Applications

PETSc has been used for modeling in all of these areas:
Acoustics, Aerodynamics, Air Pollution, Arterial Flow, Bone Fractures, Brain Surgery, Cancer Treatment, Carbon Sequestration, Cardiology, Cells, CFD, Combustion, Concrete, Corrosion, Data Mining, Dentistry, Earth Quakes, Economics, Fission, Fusion, Glaciers, Linguistics, Mantle convection ...

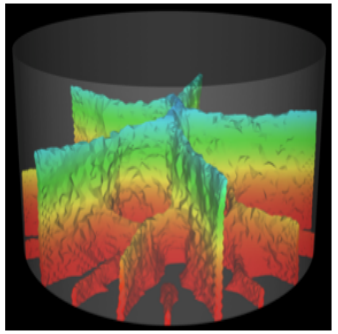
Real-time surgery



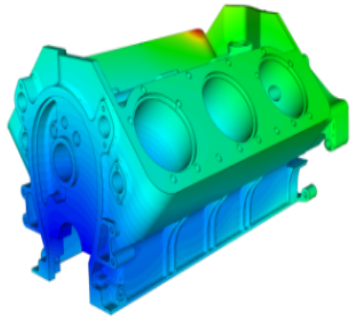
Magma dynamics



Fracture mechanics



Structure Mechanics



PETSc interfaces...

- **Dense linear algebra:** BLAS, LAPACK, Elemental
- **Sparse direct lin. sys. solvers:** MUMPS, SuperLU, SuperLU_Dist, PaStiX, UMFPACK, KLU, LUSOL, IBM ESSL, CHOLMOD, MKL_PARDISO
- **Iterative solvers / multigrid / preconditioners:** HYPRE, Trilinos ML, SPAI
- **Graph partitioning:** ParMetis, PT-Scotch, Party, Chaco
- **FFT:** FFTW
- **ODE:** Sundials
- **Data exchange:** HDF5
- **Mathematics packages:** MATLAB, Mathematica
- ...

PETSc is interfaced by ...

- **SLEPc** - Scalable Library for Eigenvalue Problems
- **FEniCS** - sophisticated Python based FEM simulation package
- **Firedrake** - partly based on FEniCS but more coupled with PETSc (mesh implementation)
- **DEAL.II** - sophisticated C++ based FEM simulation package
- **FreeFEM++** - FEM solver with embedded domain specific language
- **fluidity** - a finite element/volume fluids code
- **libMesh** - adaptive finite element library
- **MOOSE** - Multiphysics Object-Oriented Simulation Environment
- **Chaste** - Cancer, Heart and Soft Tissue Environment
- **PetIGA** - A framework for high performance Isogeometric Analysis
- **PERMON** - quadratic programming & support vector machines
- …

Important addresses

- homepage: http://www.mcs.anl.gov/petsc/
- documentation hub: http://www.mcs.anl.gov/petsc/documentation/index.html
  - PDF Users Manual – recommended for beginners
  - Alphabetical index of functions (”Index of all manual pages”)
  - examples
- download: http://www.mcs.anl.gov/petsc/download/
- BitBucket – git repository, issues, pull requests: https://bitbucket.org/petsc/petsc/

Hello World

What headers to include?

- You can include **all PETSc headers** at once by
```
#include "petsc.h" /* includes all PETSc headers */
```
- Or you can include **specific headers**
```
#include "petscsys.h" /* framework routines */
#include "petscvec.h" /* vectors */
#include "petscmat.h" /* matrices */
```
- **Higher** level headers **include** all **lower** level headers needed
```
#include "petscksp.h" /* includes vec,mat,dm,pc */
```

Initialize & Finalize (1)

```
static char help[] = "Empty program.\n\n";
#include <petscsys.h>
int main(int argc,char **argv)
{
  PetscErrorCode ierr;
  ierr = PetscInitialize(&argc,&argv,(char *)0,help);CHKERRQ(ierr);
  ierr = PetscFinalize();
  return ierr;
}
```

- Every PETSc program begins with the call to `PetscInitialize()`
- ends with the call to `PetscFinalize()`
- they call `MPI_Init()`, `MPI_Finalize()`

Initialize & Finalize (2)

```
static char help[] = "Empty program.\n\n";
#include <petscsys.h>
int main(int argc,char **argv)
{
  PetscErrorCode ierr;
  ierr = PetscInitialize(&argc,&argv,NULL,help);CHKERRQ(ierr);
  ierr = PetscFinalize();
  return ierr;
}
```

- `argc,argv` - pointer to **command line arg count** and **array**, respectively; PETSc filters out and parses PETSc options
- `help` - additional **help message** to print when run with `-help` option, or `NULL`
- handling of PETSc options will be discussed later

Communicators

- **communicator** (in MPI) = an opaque object of `MPI_Comm` type that defines **process group** and **synchronization channel**
- PETSc built-in communicators:
  - `PETSC_COMM_SELF` ⇒ just this process ⇒ for serial objects
  - `PETSC_COMM_WORLD` ⇒ all processes ⇒ for parallel objects
  - nothing special about them, you can use your own communicators
- MPI can split communicators, spawn processes on new communicators
  - PETSc does not deal with that

Error handling

- PETSc is written in C
- C has **no support** for **exceptions** (it's a C++ feature)
- instead of throwing exception, every routine returns **integer error code** (`PetscErrorCode` type)
- similarly to MPI
- error code is **„catched“ by** `CHKERRQ` **macro**

```
PetscErrorCode ierr;
ierr = SomePetscRoutine();CHKERRQ(ierr);
```

Utility routines (1)

- PETSc provides many useful utilities
- prefixed by `Petsc`
- **parallel flow control:**
  `PetscBarrier`
  `PetscSequentialPhaseBegin/End`
- **memory management and checking:**
  `PetscMalloc/Free/MallocValidate/MallocDump`

Utility routines (2)

- **logging:**
  `PetscLogEventRegister/Begin/End`
  `PetscLogStageRegister/Push/Pop`
- **string handling:**
  `PetscStrcat/cmp/cpy/len/tolower/replace/ToArray`
  `PetscSNPrintf`
- **MATLAB engine interface:**
  `PetscMatlabEngineCreate/Destroy/Evaluate`
- and **many more**

Primitive datatypes

- PETSc provides its own **primitive data types**

```
PetscInt n = 20;
PetscReal x = 2.55, y = 1e-9;
PetscComplex z = 1. + 2.*PETSC_i;
```

- most often `PetscScalar` is used which is `PetscReal/PetscComplex` based on configuration
- It is better to use them instead of **built-in C types**

$\Rightarrow$ better **portability**

$\Rightarrow$ easy switching between **real** and **complex scalars**

$\Rightarrow$ easy switching between 32-bit and 64-bit **integers**, 16-, 32-, 64-, 128-bit **real numbers**

Options database

- PETSc provides routines for managing the **options database**
- option `-help` prints help to all built-in options relevant for a given program
- in **command-line**

```
./myapp -myint 10 -myscalar 1e3 -mystring hello
```

- in **source code** `myapp.c`:

```
PetscInt  myint; PetscScalar myscalar; char mystring[1024];
PetscBool set;
PetscOptionsGetInt(NULL,NULL,"-myint",&myint,&set);
PetscOptionsGetScalar(NULL,NULL,"-myscalar",&myscalar,&set);
PetscOptionsGetString(NULL,NULL,"-mystring",mystring,1024,&set);
/* myint=10, myscalar=1e3, mystring="hello", set=PETSC_TRUE */
```

Ways to set options

1. file specified by the third argument of `PetscInitialize()`
2. files `~/.petscrc, $PWD/.petscrc, $PWD/petscrc`
   (unless `-skip_petscrc` in the file above)
3. environment variable `PETSC_OPTIONS`
4. RC file specified with command line option `-options_file [file]`
5. `PetscOptionsInsertString();` `/*` application specific hard-wired options `*/`
6. command line

- the higher number the higher priority
- in RC files, users can specify an alias for any option name
  `alias -new_name -some_long_old_option_name`

Print to standard output

`PetscPrintf`

- prints to standard output only from the **zeroth rank** in the communicator comm

C

```
PetscErrorCode PetscPrintf(MPI_Comm, const char format[], ...);
```

F

```
PetscPrintf(MPI_Comm, character(*), PetscErrorCode)
```

  - only single string can be passed

Synchronized print

To obtain output from **all processors** in **one-after-another way**, one can call:

```
PetscSynchronizedPrintf(PETSC_COMM_WORLD,
      "Hello World from %d\n",rank);
PetscSynchronizedFlush(PETSC_COMM_WORLD,PETSC_STDOUT);
```

Output:

```
Hello World from 0
Hello World from 1
Hello World from 2
```

PETSc Hello world in F

```
program main
    integer ierr, rank
#include "include/finclude/petsc.h"
    call PetscInitialize(PETSC_NULL_CHARACTER, ierr)
    call MPI_Comm_rank(PETSC_COMM_WORLD, rank, ierr)
    if (rank .eq. 0) then
        print *, ‘Hello World from ’, rank
    endif
    call PetscFinalize(ierr)
    end
```

PETSc Hello world in C

```
static char help[] = "Hello world program.\n\n";
#include <petscsys.h>
int main(int argc,char **argv)
{
  PetscErrorCode ierr;
  PetscMPIInt    rank;

  PetscInitialize(&argc,&argv,NULL,help);
  MPI_Comm_rank(PETSC_COMM_WORLD,&rank);
  PetscPrintf(PETSC_COMM_SELF,"Hello World from %d\n",rank);
  PetscFinalize();
  return 0;
}
```

PETSc Hello world in C
- with error checking

```
static char help[] = "Hello world program.\n\n";
#include <petscsys.h>
int main(int argc,char **argv)
{
  PetscErrorCode ierr;
  PetscMPIInt    rank;

  ierr = PetscInitialize(&argc,&argv,NULL,help);CHKERRQ(ierr);
  ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank);CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_SELF,"Hello World from %d\n",rank);CHKERRQ(ierr);
  ierr = PetscFinalize();
  return ierr;
}
```

Red heading will always mean hands-on

- i.e. wake yourself and your computer ☺
- look at documentation pages now
- afterwards, let's do hands-on setup – see petsc_00_setup.pdf

Thanks for your attention.