PETSc
Portable, Extensible Toolkit for Scientific Computation

Karl Rupp
rupp@mcs.anl.gov

Mathematics and Computer Science Division
Argonne National Laboratory

Tutorial at Segundo Encuentro Nacional de Computación de Alto Rendimiento para Aplicaciones Científicas

May 7th, 2013
Introducing Myself

Education

Master’s Degrees in Microelectronics and Mathematics
Doctoral Degree in Microelectronics
Home University: TU Wien

Interests

Efficient Numerics on Modern Hardware
High-level APIs
Semiconductor Device Simulation

Contact

Email: rupp@mcs.anl.gov
Web: http://www.karlrupp.net/
Find me at: Google+, Twitter, LinkedIn
Goal of this Workshop

You should learn new things about HPC

Ask Questions

Tell me if you do not understand
Ask for further details
Don’t be shy
PETSc Overview
PETSc was developed as a Platform for Experimentation

We want to experiment with different
- Models
- Discretizations
- Solvers
- Algorithms

These boundaries are often blurred...
**PETSc**

**Portable** Extensible Toolkit for Scientific Computing

**Architecture**
- tightly coupled (e.g. XT5, BG/P, Earth Simulator)
- loosely coupled such as network of workstations
- GPU clusters (many vector and sparse matrix kernels)

**Software Environment**
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Real/complex, single/double/quad precision, 32/64-bit int

**System Size**
- 500B unknowns, 75% weak scalability on Jaguar (225k cores) and Jugene (295k cores)
- Same code runs performantly on a laptop

**Free to everyone (BSD-style license), open development**
Portable **Extensible** Toolkit for Scientific Computing

**Philosophy: Everything has a plugin architecture**

Vectors, Matrices, Coloring/ordering/partitioning algorithms
Preconditioners, Krylov accelerators
Nonlinear solvers, Time integrators
Spatial discretizations/topology

**Example**

Vendor supplies matrix format and associated preconditioner, distributes compiled shared library.
Application user loads plugin at runtime, no source code in sight.
Portable Extensible **Toolkit** for Scientific Computing

**Toolset**
- algorithms
- (parallel) debugging aids
- low-overhead profiling

**Composability**
- try new algorithms by choosing from product space
- composing existing algorithms (multilevel, domain decomposition, splitting)

**Experimentation**
- Impossible to pick the solver *a priori*
- PETSc’s response: expose an algebra of composition
  - keep solvers decoupled from physics and discretization
Portable Extensible Toolkit for **Scientific Computing**

**Computational Scientists**
- PyLith (CIG), Underworld (Monash), Magma Dynamics (LDEO, Columbia), PFLOTRAN (DOE), SHARP/UNIC (DOE)

**Algorithm Developers (iterative methods and preconditioning)**

**Package Developers**
- SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM

**Funding**
- Department of Energy
  - SciDAC, ASCR ISICLLES, MICS Program, INL Reactor Program
- National Science Foundation
  - CIG, CISE, Multidisciplinary Challenge Program

**Documentation and Support**
- Hundreds of tutorial-style examples
- Hyperlinked manual, examples, and manual pages for all routines
- Support from petsc-maint@mcs.anl.gov
The Role of PETSc

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
The Role of PETSc

You want to think about how you decompose your data structures, how you think about them globally. [...] If you were building a house, you’d start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn’t start with a bunch of tiles and say. “Well I’ll put this tile down on the ground, and then I’ll find a tile to go next to it.”

But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you’re going to survive making your code parallel.

— Bill Gropp
Obtaining PETSc

Linux Package Managers
Web: http://mcs.anl.gov/petsc, download tarball
Git: https://bitbucket.org/petsc/petsc
Mercurial: https://bitbucket.org/petsc/petsc-hg

Installing PETSc

```
$> cd /path/to/petsc/workdir
$> git clone \
    https://bitbucket.org/petsc/petsc.git \
    --branch master --depth 1
$> cd petsc

$> export PETSC_DIR=$PWD PETSC_ARCH=mpich-gcc-dbg
$> ./configure --with-cc=gcc --with-fc=gfortran \
    --download-f=blas-lapack \
    --download-mpich,ml,hypre
```
Most packages can be automatically
  Downloaded
  Configured and Built (in \$PETSC_DIR/externalpackages)
  Installed with PETSc

Currently works for
  petsc4py
  PETSc documentation utilities (Sowing, lgrind, c2html)
  BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  MPICH, MPE, OpenMPI
  ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
  MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
  PaStiX, BLOPEX, FFTW, SPRNG
  Prometheus, HYPRE, ML, SPAI
  Sundials
  Triangle, TetGen, FIAT, FFC, Generator
  HDF5, Boost
PETSc Pyramid

PETSc Structure

- **PETSc PDE Application Codes**
  - ODE Integrators
  - Visualization
  - Nonlinear Solvers
  - Interface
  - Linear Solvers
  - Preconditioners + Krylov Methods
  - Object-Oriented Matrices, Vectors, Indices
  - Grid Management
  - Profiling Interface
- **Computation and Communication Kernels**
  - MPI, MPI-IO, BLAS, LAPACK
Flow Control for a PETSc Application

Main Routine

- Timestepping Solvers (TS)
- Nonlinear Solvers (SNES)
- Linear Solvers (KSP)
- Preconditioners (PC)
- Application Initialization
- Function Evaluation
- Jacobian Evaluation
- Postprocessing
Sample Code

```c
Mat A;
PetscInt m,n,M,N;
MatCreate(comm,&A);
MatSetSizes(A,m,n,M,N);   /* or PETSC_DECIDE */
MatSetOptionsPrefix(A,"foo_");
MatSetFromOptions(A);
    /* Use A */
MatView(A,PETSC_VIEWER_DRAW_WORLD);
MatDestroy(A);
```

Remarks

Mat is an opaque object (pointer to incomplete type)
Assignment, comparison, etc, are cheap
What’s up with this “Options” stuff?
We will discuss this later...
Basic PetscObject Usage

Every object in PETSc supports a basic interface

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create()</td>
<td>create the object</td>
</tr>
<tr>
<td>Get/SetName()</td>
<td>name the object</td>
</tr>
<tr>
<td>Get/SetType()</td>
<td>set the implementation type</td>
</tr>
<tr>
<td>Get/SetOptionsPrefix()</td>
<td>set the prefix for all options</td>
</tr>
<tr>
<td>SetFromOptions()</td>
<td>customize object from command line</td>
</tr>
<tr>
<td>SetUp()</td>
<td>perform other initialization</td>
</tr>
<tr>
<td>View()</td>
<td>view the object</td>
</tr>
<tr>
<td>Destroy()</td>
<td>cleanup object allocation</td>
</tr>
</tbody>
</table>

Also, all objects support the \(-\text{help}\) option.
Ways to set options

Command line

Filename in the third argument of \texttt{PetscInitialize()}

\texttt{~/petsrcrc}

\texttt{$PWD$/petsrcrc}

\texttt{$PWD/petsrcrc}

\texttt{PetscOptionsInsertFile()}

\texttt{PetscOptionsInsertString()}

\texttt{PETSC\_OPTIONS} \texttt{environment variable}

\texttt{command line option} \texttt{-options\_file [file]}
Example of Command Line Control

$> ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7
   -snes_monitor -{ksp,snes}_converged_reason
   -snes_view

$> ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7
   -snes_monitor -{ksp,snes}_converged_reason
   -snes_view -mat_view_draw -draw_pause 0.5

$> ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7
   -snes_monitor -{ksp,snes}_converged_reason
   -snes_view -mat_view_draw -draw_pause 0.5
   -pc_type lu -pc_factor_mat_ordering_type natural

Use \texttt{-help} to find other ordering types
Application Integration
Be willing to experiment with algorithms
   No optimality without interplay between physics and algorithmics

Adopt flexible, extensible programming
   Algorithms and data structures not hardwired

Be willing to play with the real code
   Toy models have limited usefulness
   But make test cases that run quickly

If possible, profile before integration
   Automatic in PETSc
Incorporating PETSc into Existing Codes

PETSc does not seize `main()`, does not control output

Propagates errors from underlying packages, flexible

Nothing special about `MPI_COMM_WORLD`

Can wrap existing data structures/algorithms

- `MatShell`, `PCShell`, full implementations
- `VecCreateMPIWithArray()`
- `MatCreateSeqAIJWithArrays()`

Use an existing semi-implicit solver as a preconditioner

Usually worthwhile to use native PETSc data structures unless you have a good reason not to

Uniform interfaces across languages

- C, C++, Fortran 77/90, Python, MATLAB

Do not have to use high level interfaces (e.g. SNES, TS, DM)

but PETSc can offer more if you do, like MFFD and SNES Test
Version Control
   It is impossible to overemphasize

Initialization
   Linking to PETSc

Profiling
   Profile before changing
   Also incorporate command line processing

Linear Algebra
   First PETSc data structures

Solvers
   Very easy after linear algebra is integrated
PETSc and Accelerators
Typical PETSc Operations

“Sparse” Linear Algebra

Sparse Matrix-Vector Operations (on-node)
Vector Operations (on and across nodes)
Only on small patches: Dense Operations (small matrices)

Arithmetic Intensity

\[ O(N) \quad O(\log(N)) \quad O(1) \]

\[ \text{Particle Methods} \quad \text{FFTs} \quad \text{Dense Linear Algebra (BLAS3)} \quad \text{SpMV, BLAS1,2} \quad \text{Lattice Methods} \quad \text{Stencils (PDEs)} \]

← Look at FLOPs Look at Mem-BW →
Computing Architecture Schematic
Good for large FLOP-intensive tasks, high memory bandwidth
PCI-Express can be a bottleneck

⇒ 10-fold speedups (usually) not backed by hardware
(applet on architecture trends)

http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html
GPU Programming Approaches

CUDA
Almost no additional code required
Vendor-lock
Relies on \texttt{nvcc} being available

OpenCL
Additional boilerplate code required (low-level API)
Broad hardware support (separate SDKs)
No more development effort from NVIDIA

Directives
Annotate existing code with OpenMP-style Pragmas
OpenACC and others
PETSc GPU Support

NVIDIA Cusp/Thrust/CUSPARSE

Compile PETSc with CUDA support
Use command line options to enable types, e.g.

```
-vec_type cusp -mat_type aijcusp
```

ViennaCL (OpenCL)

Compile PETSc with OpenCL support
Use command line options to enable types, e.g.

```
-vec_type viennacl -mat_type aijviennacl
```

Used for subsequent benchmarks

No change in application code required!
Which Accelerator is Right for Me?

Available Accelerators (Rough Sketch)

<table>
<thead>
<tr>
<th>Name</th>
<th>TFLOP/s</th>
<th>RAM (GB)</th>
<th>GB/s</th>
<th>TDP</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA GTX 580</td>
<td>1.5/∼0.2</td>
<td>1.5-3.0</td>
<td>192</td>
<td>244</td>
<td>$500</td>
</tr>
<tr>
<td>NVIDIA GTX Titan</td>
<td>4.5/1.3</td>
<td>6.0</td>
<td>288</td>
<td>250</td>
<td>$∼1k</td>
</tr>
<tr>
<td>NVIDIA Tesla 2050</td>
<td>1.3/0.5</td>
<td>3.0-6.0</td>
<td>150</td>
<td>225</td>
<td>$∼2k</td>
</tr>
<tr>
<td>NVIDIA K20</td>
<td>3.5/1.2</td>
<td>5.0</td>
<td>200</td>
<td>220</td>
<td>$∼3k</td>
</tr>
<tr>
<td>AMD HD 7970</td>
<td>3.5/∼0.9</td>
<td>3.0-6.0</td>
<td>264</td>
<td>250</td>
<td>$550</td>
</tr>
<tr>
<td>AMD FirePro W9k</td>
<td>4.0/1.0</td>
<td>6.0</td>
<td>264</td>
<td>274</td>
<td>$∼3k</td>
</tr>
<tr>
<td>Intel Xeon Phi</td>
<td>∼2.0/∼1.0</td>
<td>8</td>
<td>320</td>
<td>225</td>
<td>$∼3k</td>
</tr>
<tr>
<td>Intel Xeon E5-264x</td>
<td>0.2/0.1</td>
<td>∼64</td>
<td>∼48</td>
<td>100</td>
<td>$∼1k</td>
</tr>
</tbody>
</table>

PETSc Considerations

- Single precision performance doesn’t matter
- Essentially all kernels memory bandwidth limited
- Memory access patterns rather irregular
**Benchmarks**

Vector Addition $x = y + z$

- NVIDIA GTX 285, CUDA
- NVIDIA GTX 285, OpenCL
- AMD Radeon HD 7970, OpenCL
- Intel Xeon Phi Beta, OpenCL
- Intel Xeon Phi Beta, native
- Intel Xeon X5550, OpenMP
- Intel Xeon X5550, single-threaded
50 CG Iterations (2D FD for Poisson)

Execution Time (sec) vs. Number of Unknowns

- NVIDIA GTX 285, CUDA
- NVIDIA GTX 285, OpenCL
- AMD Radeon HD 7970, OpenCL
- Intel Xeon Phi Beta, OpenCL
- Intel Xeon Phi Beta, native
- Intel Xeon X5550, OpenMP
- Intel Xeon X5550, single-threaded
PETSc can help You

- solve algebraic and DAE problems in your application area
- rapidly develop efficient parallel code, can start from examples
- develop new solution methods and data structures
- debug and analyze performance
- advice on software design, solution algorithms, and performance

You can help PETSc

- report bugs and inconsistencies, or if you think there is a better way
tell us if the documentation is inconsistent or unclear
- consider developing new algebraic methods as plugins, contribute if your idea works

petsc-{users,dev,maint}@mcs.anl.gov