PDE Simulation with Python

Matthew Knepley
PETSc Developer
Mathematics and Computer Science Division
Argonne National Laboratory

Organization

- PETSc Introduction
- Structural Considerations
- Implementations
- Future Directions

What is PETSc?

- A freely available and supported research code
  - Download from http://www.mcs.anl.gov/petsc
  - Free for everyone, including industrial users
  - Hyperlinked manual, examples, and manual pages for all routines
  - Hundreds of tutorial-style examples
  - Support via email: petsc-maint@mcs.anl.gov
  - Usable from C, C++, Fortran 77/90, and Python

What is PETSc?

- Portable to any parallel system supporting MPI, including:
  - Tightly coupled systems

    Cray T3E, SGI Origin, IBM SP, HP 9000, Sub Enterprise
  - Loosely coupled systems, such as networks of workstations

    Compaq,HP, IBM, SGI, Sun, PCs running Linux or Windows
- PETSc History
  - Begun September 1991
  - Over 8,500 downloads since 1995 (version 2), currently 250 per month
- PETSc Funding and Support
  - Department of Energy

    SciDAC, MICS Program
  - National Science Foundation

    CIG, CISE, Multidisciplinary Challenge Program

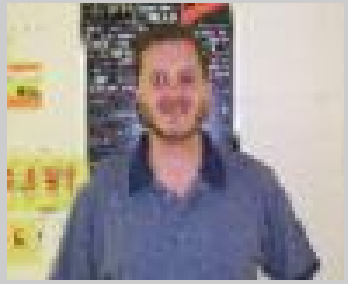
The PETSc Team


Bill Gropp


Barry Smith


Satish Balay


Dinesh Kaushik


Kris Buschelman


Matt Knepley


Hong Zhang


Victor Eijkhout


Lois McInnes

Who Uses PETSc?

- Computational Scientists
  - PyLith (TECTON), Underworld, Columbia group
- Algorithm Developers
  - Iterative methods researchers
- Package Developers
  - SLEPc, TAO, MagPar, StGermain

Automatic Downloads

- Starting in 2.2.1, some packages are automatically
  - Downloaded
  - Configured and Built (in `$PETSC_DIR/externalpackages`)
  - Installed in PETSc
- Currently works for
  - PETSc documentation utilities (Sowing, lgrind, c2html)
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, LAM
  - ParMetis, Chaco, Jostle, Party, Scotch
  - MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack
  - Prometheus, HYPRE, ML, SPAI
  - Sundials

Structure of Scalable PDE Algorithms

Hierarchy

The central feature of algorithms for PDEs is

hierarchical decomposition

- Key operations
  - Restriction
  - Assembly
- Bulk of the computation is local

http://www.mcs.anl.gov/petsc/petsc-as/documentation/tutorials/sieve.pdf

Algorithms

- Multigrid
- FMM
- Finite elements
- Finite difference
- PETSc DA parallelism

Implications

Python for control and logic

C for local computation

- Decouple organization of storage from mathematical operations
  - Vectors are **not** arrays
- Lots of small arrays
  - `get/setValues()` methods
- Views into larger arrays
- Dense, local computation is cache/bandwidth efficient

Wrapper Implementations

Implementations

- PETSc (SIDL)
  - Parsers a SIDL interface description
  - Generates a C extension module for each class and Python for enums
- petsc4py (SWIG)
  - Lisandro Dalcin (CIMEC)
  - Parsers headers
  - Generates a single C extension module and Python for each class
- pypetsc (Pyrex)
  - Simon Burton (ANU)
  - Parsers headers
  - Generates a single C extensions module and Python infrastructure
  - Python handles initialization/finalization and creation/destruction

Commonalities

Interface is more important than implementation

- Class and method names
  - Type signatures
  - Function pointer signatures
- Enumerations
- Static factory methods
  - Not in SIDL
- Basically the SIDL

Cool Things

- Multiple import roots
  - Necessary for componentized development
- Function pointers (closures)
  - Dispatch from a suitable C wrapper
  - Alternative to interfaces (SIDL)
  - Type checking is dynamic (an exception thrown on arg mismatch)

Multiple Import Roots

- Hooks Modifications
  - Augment the default search path
- Loader Modifications
  - **find_module()** now returns a list of paths
- Importer Modifications
  - Module **__path__** member is now a list
  - Install custom loader and importer
- Installation

```
loader   = Loader(Hooks())
importer = Importer(loader)
importer.install()
```

- Code in `$PETSC_DIR/python/sidl/BuildSystem/importer.py`

Function Pointers

- Input is parsed as a object pointer in the C API
- A C dispatch function is set as the callback
  - The object pointer becomes the context arg
- The dispatch function calls the method with appropriate args
  - Of course, no context arg (come from lexical scope)

Distributing the Wrappers

- All implementations distribute
  - C that links to the Python library and PETSc
  - Some Python
- SIDL version is currently in PETSc
  - Configure with `-with_python -with_shared -with_dynamic`
  - Other versions will be released this fall (distutils)
- Harder to distribute construction mechanism

Examples

- 2D Poisson
  - Finite Differences
  - `$PETSC_DIR/src/snes/examples/tutorials/ex1.py`
- 2D Bratu
  - $-\Delta u - \lambda e^u = 0$
  - Finite Differences
  - `$PETSC_DIR/src/snes/examples/tutorials/ex2.py`
- 2D Poisson
  - P_1 finite elements
  - `petsc4py/test/test_poisson2.py`

The Future

What I Would Do Differently

- Allow Python to handle:
  - Object structure
  - Dynamic loading
- Automate double dispatch
  - Could allow mixing precisions
- Better semantics
  - Specify who is responsible for memory

Shameless Plugs

FENICS Project:

http://www.fenics.org

- FInite element Automatic Tabulator
  - Declarative specification of elements
  - Library of quadratures and shapes
  - Generates complete discrete jet of an element
- Fenics Form Compiler
  - Constructs element tensors from weak forms
  - Generates source code
  - Uses FIAT and exact quadrature