High Performance Python Libraries

Matthew Knepley

Computation Institute
University of Chicago
Department of Mathematics and Computer Science
Argonne National Laboratory

4th Workshop on Python for High Performance and Scientific Computing (PyHPC)
SC14: New Orleans, LA November 17, 2014
New Model for Scientific Software

- sympy
- symbolics
- eqn. definition
- FFC/SyFi
- data structures
- integration/assembly
- solvers
- numpy
- petsc4py
- PyCUDA
- PETSc
- CUDA OpenCL

Figure: Schematic for a generic scientific application
Conservation Laws Package:

- Solves general hyperbolic PDEs in 1/2/3 dimensions
- Developed by many authors over 20 years in Fortran 77
- Dozens of contributed Riemann solvers
- Textbook and many examples available
PyClaw

- Python interface to Clawpack
- Easy parameter studies and numerical experiments
- Strong focus on reproducible research
- Leverage interface to matplotlib
- Pure python based version of Clawpack

See David Ketcheson’s Slides
PyClaw

- Python interface to Clawpack
- Easy parameter studies and numerical experiments
- Strong focus on reproducible research
- Leverage interface to matplotlib
- Pure python based version of Clawpack

See David Ketcheson’s Slides
PyClaw

Architecture

- PyClaw
- numpy
- Fortran
- C
- Python
- Python Extension
- petsc4py
- PETSc
- MPI
- Clawpack kernels

M. Knepley (UC)
Changes to PyClaw (less than 300 LOC):

- Store grid data in DMDA instead of NumPy array
- Calculate global CFL condition by reduction
- Update neighbor information after successful time steps
  - Through grid.q property

Both the top and bottom level code components are purely serial
Only change to user code:

```python
if use_petsc:
    import clawpack.petclaw as pyclaw
else:
    from clawpack import pyclaw
```
PyClaw
Weak Scaling for Euler Equations

![Graph showing execution time per core for different numbers of cores (1, 16, 256, 4096, 65536).]

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>CFL Reduce</th>
<th>Parallel Initialization</th>
<th>Global to Local</th>
<th>Local to Global</th>
<th>Serial Computations</th>
<th>Total Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.001</td>
<td>2</td>
<td>5.9</td>
<td>4.9</td>
<td>88.2</td>
<td>1.0</td>
</tr>
<tr>
<td>16</td>
<td>4.1</td>
<td>1.5</td>
<td>3.6</td>
<td>2.5</td>
<td>92.7</td>
<td>0.97</td>
</tr>
<tr>
<td>256</td>
<td>3.9</td>
<td>1.5</td>
<td>3.6</td>
<td>2.5</td>
<td>92.6</td>
<td>0.97</td>
</tr>
<tr>
<td>4096</td>
<td>3.8</td>
<td>1.9</td>
<td>3.6</td>
<td>2.5</td>
<td>92.7</td>
<td>0.97</td>
</tr>
<tr>
<td>65536</td>
<td>4.2</td>
<td>6.6</td>
<td>3.6</td>
<td>2.5</td>
<td>92.8</td>
<td>0.92</td>
</tr>
</tbody>
</table>
Reproducibility Repository (Aron Ahmadia)

https://bitbucket.org/ahmadia/pyclaw-sisc-rr
Interactive Demos from Paper

Python reproducibility tools far more advanced than C counterparts
PyWENO, from Matthew Emmett

- Computes arbitrary order 1D WENO reconstructions
- Generates Fortran, C, and OpenCL kernels on the fly
- Problem domain is completely mathematically specified
Succeeds by combining mature packages

Clawpack and SharpClaw
- Provide computational kernels for time-dependent nonlinear wave propagation

PETSc and petsc4py
- Manage distributed data, parallel communication, linear algebra, and elliptic solvers

numpy and f2py
- Provide array API for data communication and wrappers
1. PyClaw
2. PyLith
3. FEniCS
PyLith

- Multiple problems
  - Dynamic rupture
  - Quasi-static relaxation
- Multiple models
  - Nonlinear visco-plastic
  - Finite deformation
  - Fault constitutive models
- Multiple meshes
  - 1D, 2D, 3D
  - Hex and tet meshes
- Parallel
  - PETSc solvers
  - DMPLex mesh management

---

\(^a\) Aagaard, Knepley, Williams
Multiple Mesh Types

- Triangular
- Tetrahedral
- Rectangular
- Hexahedral
Simulation of aseismic creep along the fault between subducting oceanic crust and the lithosphere/mantle

Slip rate of 8 cm/yr.
PyLith can create complex boundary objects on the fly,

```python
[pylithapp.timedependent]
bc = [boundary_east_mantle, boundary_west, boundary_bottom_mantle]

[pylithapp.timedependent.bc.boundary_east_mantle]
bc_dof = [0]
label = bndry_east_mantle
db_initial.label = Dirichlet BC on east boundary (mantle)

[pylithapp.timedependent.bc.boundary_west]
bc_dof = [0]
label = bndry_west
db_initial.label = Dirichlet BC on west boundary

[pylithapp.timedependent.bc.boundary_bottom_mantle]
bc_dof = [1]
label = bndry_bot_mantle
db_initial.label = Dirichlet BC on bottom boundary (mantle)
```
as well as faults, identified with portions of the input mesh,

```python
[pylithapp.timedependent]
interfaces = [fault_slabtop, fault_slabbot]

[pylithapp.timedependent.interfaces]
fault_slabtop = pylith.faults.FaultCohesiveKin
fault_slabbot = pylith.faults.FaultCohesiveKin

[pylithapp.timedependent.interfaces.fault_slabtop]
label = fault_slabtop
id = 100
quadrature.cell = pylith.feassemble.FIATSimplex
quadrature.cell.dimension = 1

[pylithapp.timedependent.interfaces.fault_slabtop]
label = fault_slabtop
id = 101
quadrature.cell = pylith.feassemble.FIATSimplex
quadrature.cell.dimension = 1
```
and configure a precise rupture sequence

```python
[timedependent.interfaces.fault_slabtop.eq_srcs.rupture]
slip_function = pylith.faults.ConstRateSlipFn

[timedependent.interfaces.fault_slabtop.eq_srcs.rupture.slip_function]
slip_rate.iohandler.filename = fault_creep_slabtop.spatialdb
slip_rate.query_type = linear
slip_rate.label = Final slip

slip_time = spatialdata.spatialdb.UniformDB
slip_time.label = Slip time
slip_time.values = [slip−time]
slip_time.data = [0.0*year]
```
on each fault.

```python
[timedependent.interfaces.fault_slabbot.eq_srcs.rupture]
slip_function = pylith.faults.ConstRateSlipFn

[timedependent.interfaces.fault_slabbot.eq_srcs.rupture.slip_function]
slip_rate = spatialdata.spatialdb.UniformDB
slip_rate.label = Slip rate
slip_rate.values = [left-lateral-slip, fault-opening]
slip_rate.data = [8.0*cm/year, 0.0*cm/year]

slip_time = spatialdata.spatialdb.UniformDB
slip_time.label = Slip time
slip_time.values = [slip-time]
slip_time.data = [0.0*year]
```
PyLith packages the solve, enabling numerical Green’s functions,

class GreensFns(Problem):
    def run(self, app):
        """Compute Green’s functions associated with fault slip."""
        self.checkpointTimer.toplevel = app  # Set handle for saving state
        # Limit material behavior to linear regime
        for material in self.materials.components():
            material.useElasticBehavior(True)
        nimpulses = self.source.numImpulses()
        ipulse = 0;
        dt = 1.0
        while ipulse < nimpulses:
            # Set t=ipulse−dt, so that t+dt corresponds to the impulse
            t = float(ipulse)−dt
            self.checkpointTimer.update(t)

            self.formulation.prestep(t, dt)
            self.formulation.step(t, dt)
            self.formulation.poststep(t, dt)

            ipulse += 1
# Get GF impulses and calculated responses from HDF5
(impCoords, impVals, respCoords, respVals) = getImpResp()
# Get observed displacements and observation locations.
(dataCoords, dataVals) = getData()
# Get penalty parameters.
penalties = numpy.loadtxt(penaltyFile, dtype=numpy.float64)
# Determine matrix sizes and set up A–matrix.
numParams = impVals.shape[0]
numObs = 2 * dataVals.shape[1]
aMat = respVals.reshape((numParams, numObs)).transpose()
# Create diagonal matrix to use as the penalty.
parDiag = numpy.eye(numParams, dtype=numpy.float64)
# Data vector, plus a priori parameters (assumed to be zero).
dataVec = numpy.concatenate((dataVals.flatten(), numpy.zeros(numParams)))

### Loop over number of inversions.

# Output results.
f = open(outputFile, "w")
f.write(head)
numpy.savetxt(f, invResults, fmt="%.14e")
f.close()
### Read Data and Setup

```python
for inversion in range(numInv):
    # Scale diagonal by penalty parameter, and stack
    penMat = penalty * parDiag
    designMat = numpy.vstack((aMat, penMat))
    designMatTrans = designMat.transpose()

    # Form generalized inverse matrix.
    normeq = numpy.dot(designMatTrans, designMat)
    genInv = numpy.dot(numpy.linalg.inv(normeq), designMatTrans)

    # Solution is product of generalized inverse with data vector.
    solution = numpy.dot(genInv, dataVec)
    invResults[:, 2 + inversion] = solution

    # Compute predicted results and residual.
    predicted = numpy.dot(aMat, solution)
    residual = dataVals.flatten() - predicted
    residualNorm = numpy.linalg.norm(residual)
```

### Output results.
Problem

Debugging cross language is hard

- gdb 7 adds valuable Python support
- Active development on C side means frequent refactoring

No good Python installation answer for C packages

- HPC requires testing
- HPC has more dependent packages, e.g. MPI
Problem

Debugging cross language is hard

- gdb 7 adds valuable Python support
- Active development on C side means frequent refactoring

No good Python installation answer for C packages

- HPC requires testing
- HPC has more dependent packages, e.g. MPI
Outline

1. PyClaw
2. PyLith
3. FEniCS
FEniCS allows the automated solution of differential equations by finite element methods:

- automated solution of variational problems,
- automated error control and adaptivity,
- comprehensive library of finite elements,
- high performance linear algebra.

Incredibly difficult problems coded and solved quickly
FEniCS allows the automated solution of differential equations by finite element methods:

- automated solution of variational problems,
- automated error control and adaptivity,
- comprehensive library of finite elements,
- high performance linear algebra.

Incredibly difficult problems coded and solved quickly
Topology Optimization

From Patrick E. Farrell, minimization of dissipated power in a fluid

Figure 10. Design domain for the double pipe example.
Topography Optimization

From Patrick E. Farrell, minimization of dissipated power in a fluid

\[ \frac{1}{2} \int_{\Omega} \alpha(\rho) u \cdot u + \mu \int_{\Omega} \nabla u : \nabla u - \int_{\Omega} fu \]

subject to the Stokes equations with velocity Dirichlet conditions

\[ \alpha(\rho)u - \mu \nabla^2 u + \nabla p = f \quad \text{in } \Omega \]
\[ \text{div}(u) = 0 \quad \text{on } \Omega \]
\[ u = b \quad \text{on } \delta \Omega \]

and to the control constraints on available fluid volume

\[ 0 \leq \rho(x) \leq 1 \quad \forall x \in \Omega \]
\[ \int_{\Omega} \rho \leq V \]
Topology Optimization

With variables,

\begin{align*}
    u & \quad \text{velocity} \\
    p & \quad \text{pressure} \\
    \rho & \quad \text{control} \\
    V & \quad \text{volume bound} \\
    \alpha(\rho) & \quad \text{inverse permeability}
\end{align*}

where

\[
\alpha(\rho) = \bar{\alpha} + (\alpha - \bar{\alpha}) \rho \frac{1 + q}{\rho + q}
\]

The parameter \( q \) penalizes deviations from the values 0 or 1.

FEniCS has reified functions spaces, making them easy to combine.

\[
\begin{align*}
N &= 200 \\
\text{delta} &= 1.5 \quad \text{# The domain is 1 high and delta wide} \\
V &= \text{Constant}(1.0/3) \times \text{delta} \quad \text{# fluid should occupy 1/3 of the domain} \\
\text{mesh} &= \text{RectangleMesh}(0.0, 0.0, \text{delta}, 1.0, N, N) \\
A &= \text{FunctionSpace}(<\text{mesh}>, "CG", 1) \quad \text{# control function space} \\
U &= \text{VectorFunctionSpace}(<\text{mesh}>, "CG", 2) \quad \text{# velocity function space} \\
P &= \text{FunctionSpace}(<\text{mesh}>, "CG", 1) \quad \text{# pressure function space} \\
W &= \text{MixedFunctionSpace}([U, P]) \quad \text{# Taylor–Hood function space}
\end{align*}
\]
Patrick has packaged up the forward problem, allowing adjoint solves, leading to solution of optimization problems.

```python
def forward(rho):
    """Solve the forward problem for a given fluid distribution rho(x).""
    w = Function(W)
    (u, p) = split(w)
    (v, q) = TestFunctions(W)

    F = (alpha(rho) * inner(u, v) * dx + inner(grad(u), grad(v)) * dx +
         inner(grad(p), v) * dx + inner(div(u), q) * dx)
    bc = DirichletBC(W.sub(0), InflowOutflow(), "on_boundary")
    solve(F == 0, w, bcs=bc)

    return w
```
The weak form language is reused to define cost functionals...

\[
J = \text{Functional}(0.5 \ast \text{inner}(\alpha(rho) \ast u, u) \ast dx + \\
\mu \ast \text{inner}(\text{grad}(u), \text{grad}(u)) \ast dx)
\]

\[
m = \text{SteadyParameter}(rho)
\]

\[
\text{Jhat} = \text{ReducedFunctional}(J, m, \text{eval\_cb}=\text{eval\_cb})
\]

\[
\text{rfn} = \text{ReducedFunctional\_NumPy}(\text{Jhat})
\]
... and constraints.

class VolumeConstraint ( InequalityConstraint ) :
    """ A class that enforces the volume constraint \( g(a) = V - a \cdot dx \geq 0. \"""
    def __init__(self, V):
        self.V = float(V)
        self.smass = assemble(TestFunction(A) * Constant(1) * dx)
        self.tmpvec = Function(A)

    def function(self, m):
        print "Evaluating constraint residual"
        self.tmpvec.vector()[:] = m

        # Compute the integral of the control over the domain
        integral = self.smass.inner(self.tmpvec.vector())
        print "Current control integral: ", integral
        return [self.V - integral]

    def jacobian(self, m):
        print "Computing constraint Jacobian"
        return [-self.smass]
Composibility of package interfaces

- PETSc solvers accessed through C interface inside Dolfin
- Original wrapper did not anticipate problems with bounds
- Scalable solution using SNESVI from PETSc
- New deflation algorithm (Farrell) finds all solutions
- Should refactor to use petsc4py in FEniCS
Composibilty of package interfaces

- PETSc solvers accessed through C interface inside Dolfin
- Original wrapper did not anticipate problems with bounds
- Scalable solution using SNESVI from PETSc
- New deflation algorithm (Farrell) finds all solutions
- Should refactor to use petsc4py in FEniCS
Composibility of package interfaces

- PETSc solvers accessed through C interface inside Dolfin
- Original wrapper did not anticipate problems with bounds
  - Scalable solution using SNESVI from PETSc
  - New deflation algorithm (Farrell) finds all solutions
  - Should refactor to use petsc4py in FEniCS
Composibility of package interfaces

- PETSc solvers accessed through C interface inside Dolfin
- Original wrapper did not anticipate problems with bounds
- Scalable solution using SNESVI from PETSc
  - New deflation algorithm (Farrell) finds all solutions
  - Should refactor to use petsc4py in FEniCS
Composibility of package interfaces

- PETSc solvers accessed through C interface inside Dolfin
- Original wrapper did not anticipate problems with bounds
- Scalable solution using SNESVI from PETSc
- New deflation algorithm (Farrell) finds all solutions
- Should refactor to use petsc4py in FEniCS
Composibilty of package interfaces

- PETSc solvers accessed through C interface inside Dolfin
- Original wrapper did not anticipate problems with bounds
- Scalable solution using SNESVI from PETSc
- New deflation algorithm (Farrell) finds all solutions
- Should refactor to use petsc4py in FEniCS
HPC is an enterprise driven by academia

Assessing the impact of a package is hard.

Citation counts are an imperfect measure.

HPC is an enterprise
driven by academia

Assessing the impact of a package is hard.

Citation counts are an important measure.

Accurately Citing Software and Algorithms Used in Publications,
HPC is an enterprise driven by academia

Assessing the impact of a package is hard.

Citation counts are an imperfect measure.

Accurately Citing Software and Algorithms Used in Publications,
PyClaw Impact

- 10 publications based on PyClaw
- 60 citations of PyClaw papers
- 1800+ citations of Clawpack papers
- 4992 downloads (pip) in 2014
- $\approx 3,750$ Google Hits
- GitHub: 46 forks, 44 stars, 21 contributors
PyLith Impact

- 33 publications based on PyLith
- 50+ citations of PyLith paper/abstracts
- Downloads: 30,000+
- ≈ 6000 Google Hits
- Dedicated tutorial conference every two years
27 author publications
700 citations of main papers
50,000 downloads of The FEniCS Book in 2013
≈ 205,000 Google Hits
Annual FEniCS conference ≈ 50 attendees
We need **composable** libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2
Kernel Libraries

We need **composable** libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2
We need **composable** libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2
We need **composable** libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2
We need **composable** libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2