High Performance Python Libraries

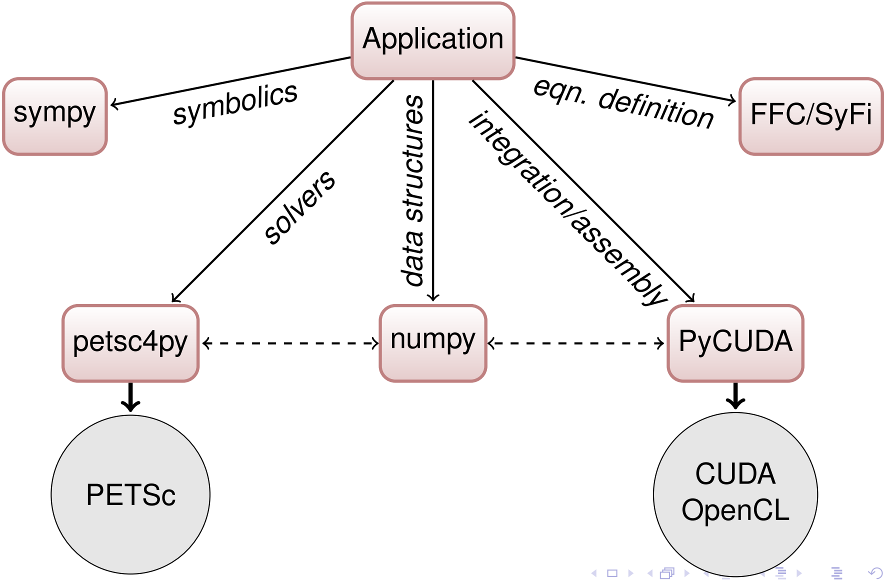
Matthew Knepley

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



Outline

1. PyClaw
2. PyLith
3. FEniCS

Clawpack

www.clawpack.org

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

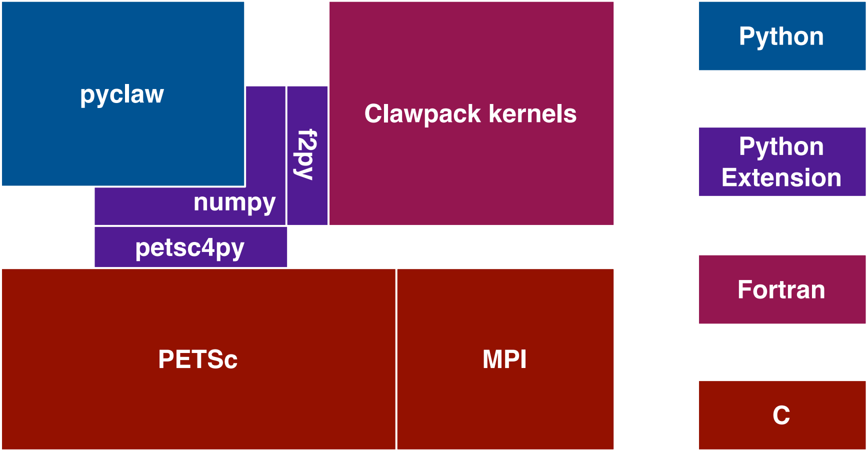
See David Ketcheson's Slides

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

Parallelism

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

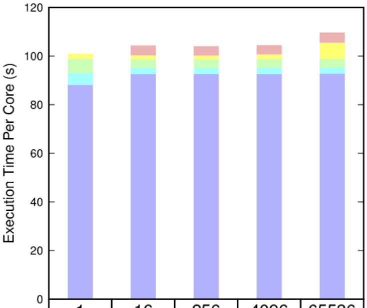
PyClaw
Parallelism

Only change to user code:

```
if use_petsc:
    import clawpack.petclaw as pyclaw
else:
    from clawpack import pyclaw
```

PyClaw

Weak Scaling for Euler Equations



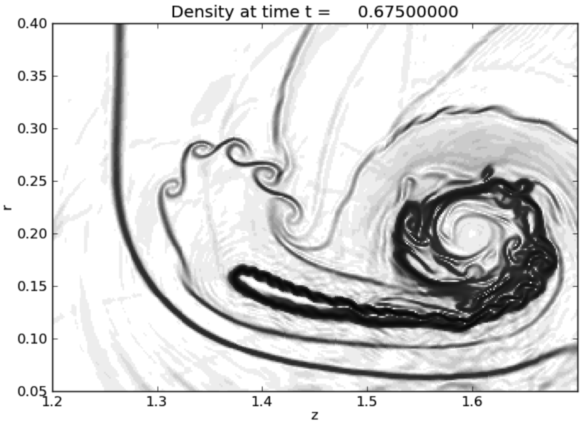
	1	16	256	4096	65536
CFL Reduce	0.001	4.1	3.9	3.8	4.2
Parallel Initialization	2	1.5	1.5	1.9	6.6
Global to Local	5.9	3.6	3.6	3.6	3.6
Local to Global	4.9	2.5	2.5	2.5	2.5
Serial Computations	88.2	92.7	92.6	92.7	92.8
Total Efficiency	1.0	0.97	0.97	0.97	0.92

Number of Cores

PyClaw
Reproducibility

Reproducibility Repository (Aron Ahmadia)

https://bitbucket.org/ahmadia/pyclaw-sisc-rr

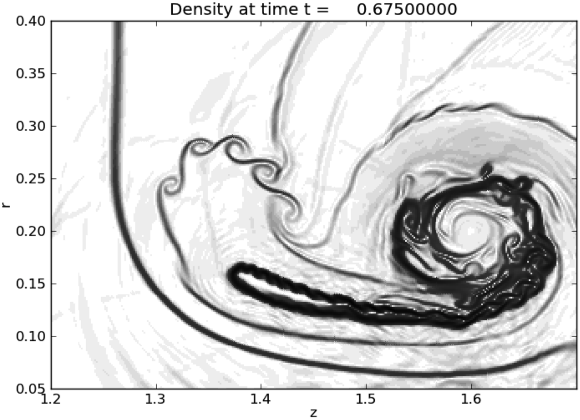


PyClaw

Reproducibility

Interactive Demos from Paper

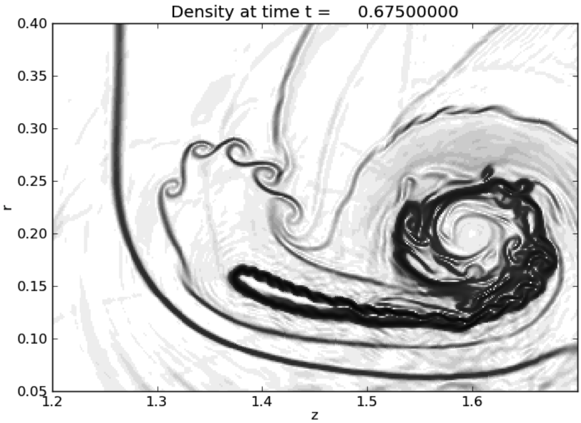
http://numerics.kaust.edu.sa/papers/pyclaw-sisc/pyclaw-sisc.html



PyClaw

Reproducibility

Python reproducibility tools far more advanced than C counterparts



PyClaw

Code Generation

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

PyClaw

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

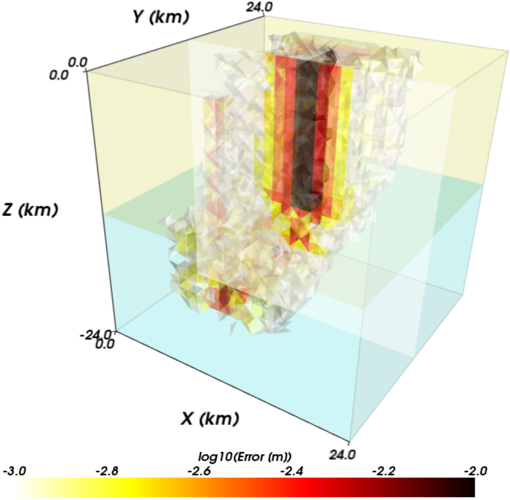
- Provide array API for data communication and wrappers

Outline

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

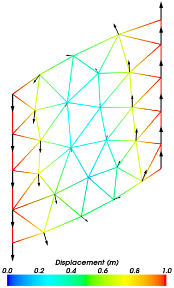


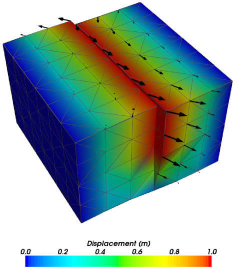
---

[a] Aagaard, Knepley, Williams

Multiple Mesh Types

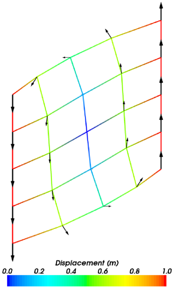
Triangular

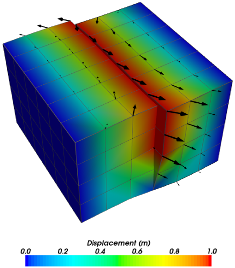




Tetrahedral

Rectangular

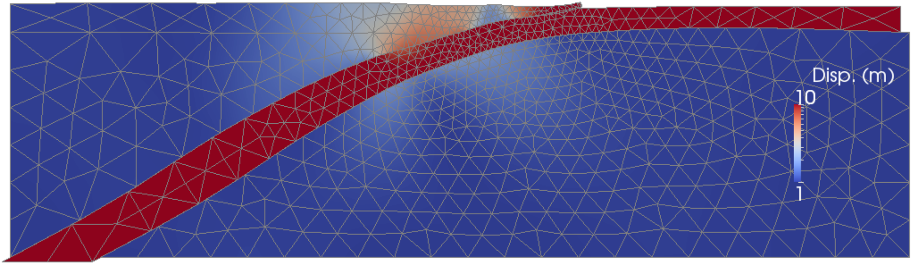




Hexahedral

PyLith
Input

Simulation of aseismic creep along the fault between subducting oceanic crust and the lithosphere/mantle



Slip rate of 8 cm/yr.

PyLith

Input

PyLith can create complex boundary objects on the fly,

```
[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)
```

PyLith

Input

as well as faults, identified with portions of the input mesh,

```
[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
```

PyLith
Input

and configure a precise rupture sequence

```
[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]
```

PyLith

Input

on each fault.

```
[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]
```

Green's Functions

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
```

Green's Functions

```
# 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="%14.6e")
f.close()
```

Green's Functions

```
### Read Data and Setup
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

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

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

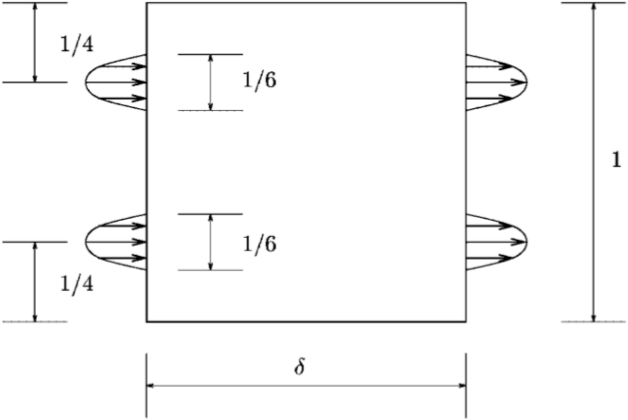


Figure 10. Design domain for the double pipe example.

Topology Optimization

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

$$\frac{1}{2}\int_{\Omega} \alpha(\rho) u \cdot u + \mu \int_{\Omega} \nabla u : \nabla u - \int_{\Omega} f u$$

subject to the Stokes equations with velocity Dirichlet conditions

$$\begin{aligned} \alpha(\rho) u - \mu \nabla^2 u + \nabla p &= f && \text{in } \Omega \\ \mathrm{div}(u) &= 0 && \text{on } \Omega \\ u &= b && \text{on } \delta\Omega \end{aligned}$$

and to the control constraints on available fluid volume

$$\begin{aligned} 0 \leq \rho(x) \leq 1 & \qquad \forall x \in \Omega \\ \int_{\Omega} \rho \leq V & \end{aligned}$$

Topology Optimization

With variables,

u	velocity
p	pressure
ρ	control
V	volume bound
$\alpha(\rho)$	inverse permeability

where

$$\alpha(\rho) = \bar{\alpha} + (\underline{\alpha} - \bar{\alpha})\rho\frac{1+q}{\rho+q}$$

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

http://dolfin-adjoint.org/documentation/stokes-topology/stokes-topology.html

Topology Optimization

Function Spaces

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

```
N     = 200
delta = 1.5                              # The domain is 1 high and delta wide
V     = Constant(1.0/3) * delta          # fluid should occupy 1/3 of the domain

mesh = RectangleMesh(0.0, 0.0, delta, 1.0, N, N)
A = FunctionSpace(mesh, "CG", 1)        # control function space
U = VectorFunctionSpace(mesh, "CG", 2)  # velocity function space
P = FunctionSpace(mesh, "CG", 1)        # pressure function space
W = MixedFunctionSpace([U, P])          # Taylor-Hood function space
```

Topology Optimization

Forward Solves

Patrick has packaged up the forward problem, allowing adjoint solves, leading to solution of optimization problems.

```
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
```

Topology Optimization

Functionals

The weak form language is reused to define cost functionals…

```
J    = Functional(0.5 * inner(alpha(rho) * u, u) * dx +
                      mu * inner(grad(u), grad(u)) * dx)
m    = SteadyParameter(rho)
Jhat = ReducedFunctional(J, m, eval_cb=eval_cb)
rfn  = ReducedFunctionalNumPy(Jhat)
```

Topology Optimization

Inverse problem

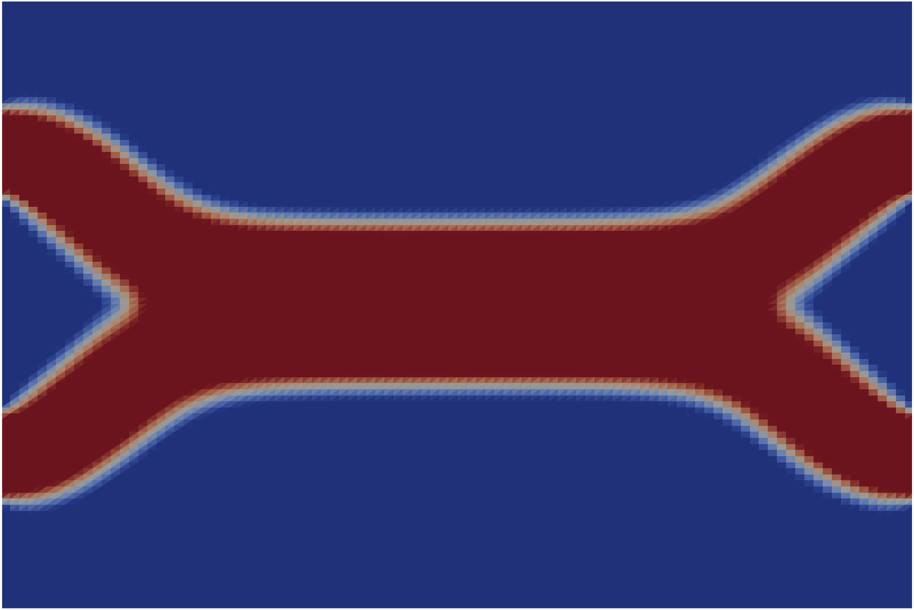
. . . and constraints.

```
class VolumeConstraint(InequalityConstraint):
  """A class that enforces the volume constraint g(a) = V − a∗dx >= 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 "Evaluting 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]
```



Problem

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

Problem

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

Problem

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

Problem

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

Problem

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

Problem

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.

Accurately Citing Software and Algorithms Used in Publications,
http://files.figshare.com/1187013/paper.pdf

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,
http://files.figshare.com/1187013/paper.pdf

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,
http://files.figshare.com/1187013/paper.pdf

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+
- $\approx$ 6000 Google Hits
- Dedicated tutorial conference every two years

FEniCS Impact

- 27 author publications
  700 citations of main papers
- 50,000 downloads of The FEniCS Book in 2013
- $\approx 205,000$ Google Hits
- Annual FEniCS conference $\approx$ 50 attendees

Kernel Libraries

We need composable libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2

Kernel Libraries

We need composable libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2

Kernel Libraries

We need composable libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2

Kernel Libraries

We need composable libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2

Kernel Libraries

We need composable libraries of kernels:

- PyClaw
- FEniCS
- PETSc
- OCCA2