

# FiPy in Parallel

SIAM CONFERENCE ON CSE, 03/01/2011

Daniel Wheeler, Jonathan E. Guyer and  
James O'Beirne

[www.ctcms.nist.gov/fipy](http://www.ctcms.nist.gov/fipy)

Metallurgy Division  
Materials Measurement Laboratory  
National Institute of Standards and Technology

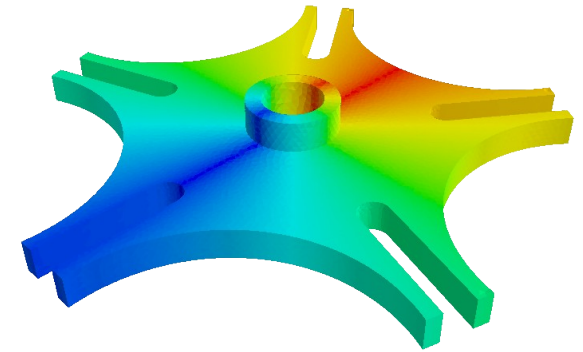
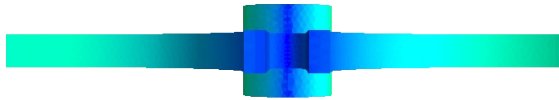
# Outline

- **FiPy example**
- FiPy overview
- Parallel choices
- Parallel design:
  - Unified communicator Interface
  - Partitioning
  - Testing
- FiPy example in parallel
- Conclusion

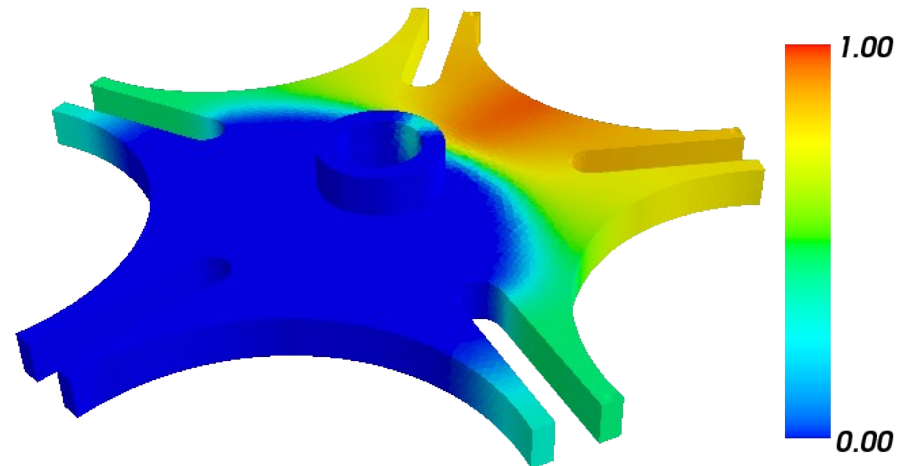


# FiPy Example

```
## piece.py
from fipy import *
m = Gmsh3D(open('piece.geo').read())
x, y, z = m.cellCenters
phi = CellVariable(mesh=m,
                   value=arccos(x / sqrt(x**2 + y**2)) / pi)
Viewer(phi).plot()
X, Y, Z = m.faceCenters
phi.faceValue.constrain(value=0,
                        where=(Z == -0.2))
eq = (TransientTerm() == DiffusionTerm(1 / (phi + 1e-3)**2) + 0.1 * phi)
for i in range(10):
    eq.solve(phi, dt=1e-3)
Viewer(phi).plot()
```



$$\frac{\partial \phi}{\partial t} = \nabla \cdot \left( \frac{1}{\phi^2} \nabla \phi \right) + 0.1 \phi$$

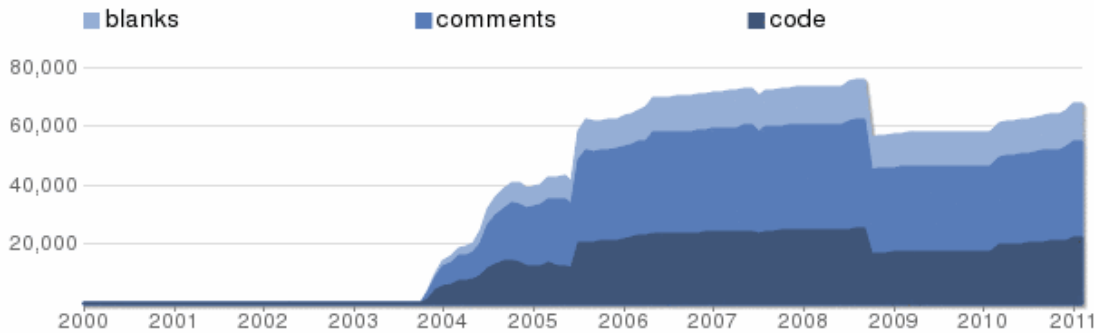


# Outline

- FiPy example
- **FiPy overview**
- Parallel choices
- Parallel design:
  - Unified communicator Interface
  - Partitioning
  - Testing
- FiPy example in parallel
- Conclusion



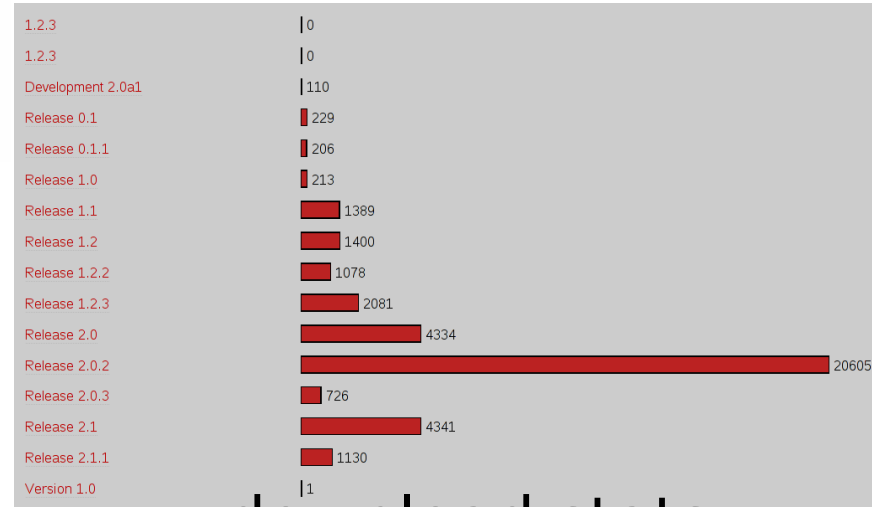
# FiPy Overview



Lines of Code By Language

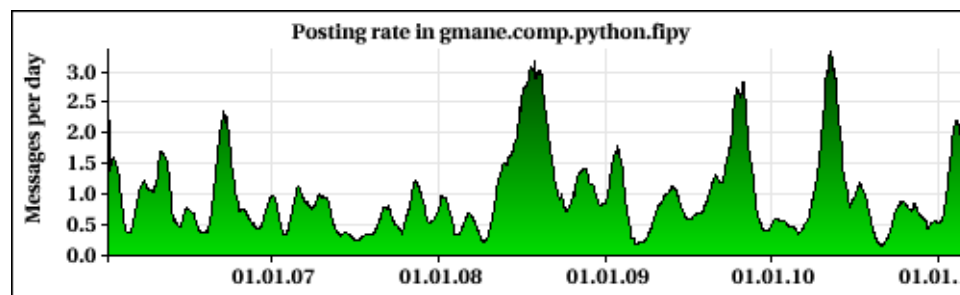
Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines
<a href="#">Python</a>	21,117	32,576	60.7%	12,599	66,292
<a href="#">CSS</a>	516	23	4.3%	114	653
<a href="#">HTML</a>	417	12	2.8%	31	460
<a href="#">TeX/LaTeX</a>	140	0	0.0%	13	153
<a href="#">JavaScript</a>	49	1	2.0%	2	52

This analysis was updated 2 days ago. (22 Feb 2011 17:12 UTC)



download stats

mailing list stats



# FiPy Overview

## Publications

### Attention

If you use FiPy in your research, please cite:

J. E. Guyer, D. Wheeler & J. A. Warren, "FiPy: Partial Differential Equations with Python," *Computing in Science & Engineering* **11** (3) pp. 6-15 (2009), doi:10.1109/MCSE.2009.52. (pdf)

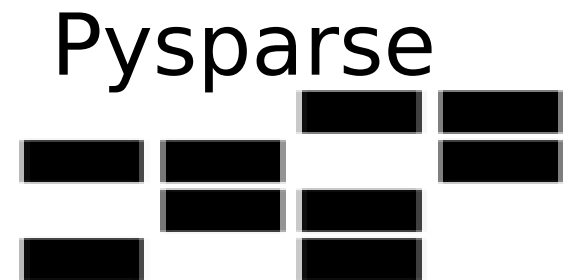
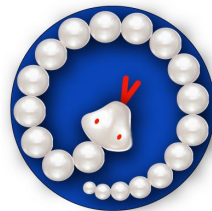
Other publications that have used FiPy. Please contact us to add your work to this list.

- D. Wheeler, and J. A. Warren & W. J. Boettinger, "Modeling the early stages of reactive wetting" *Physical Review E* **82** (5) pp. 051601 (2010), doi:10.1103/PhysRevE.82.051601
- R. R. Mohanty, J. E. Guyer & Y. H. Sohn, "Diffusion under temperature gradient: A phase-field model study" *Journal of Applied Physics* **106** (3) pp. 034912 (2009), doi:10.1063/1.3190607.
- J. A. Warren, T. Pusztai, L. Környei & L. Gránásy, "Phase field approach to heterogeneous crystal nucleation in alloys," *Physical Review B* **79** 014204 (2009), doi:10.1103/PhysRevB.79.014204.
- T. P. Moffat, D. Wheeler, S.-K. Kim & D. Josell, "Curvature enhanced adsorbate coverage mechanism for bottom-up superfilling and bump control in damascene processing," *Electrochimica Acta* **53** (1) pp. 145-154 (2007), doi:10.1016/j.electacta.2007.03.025.
- W. J. Boettinger, J. E. Guyer, C. E. Campbell, G. B. McFadden, "Computation of the Kirkendall velocity and displacement fields in a one-dimensional binary diffusion couple with a moving interface," *Proceedings of the Royal Society A: Mathematical, Physical & Engineering Sciences* **463** (2088) pp. 3347-3373 (2007), doi:10.1098/rspa.2007.1904.
- T. Cickovski, K. Aras, M. Swat, R. M. H. Merks, T. Glimm, H. G. E. Hentschel, M. S. Alber, J. A. Glazier, S. A. Newman & J. A. Izaguirre, "From Genes to Organisms Via the Cell: A Problem-Solving Environment for Multicellular Development," *Computing in Science & Engineering* **9** (4) pp. 50-60 (2007), doi:10.1109/MCSE.2007.74.
- L. Gránásy, T. Pusztai, D. Saylor & J. A. Warren, "Phase Field Theory of Heterogeneous Crystal Nucleation," *Physical Review Letters* **98** 035703 (2007) 10.1103/PhysRevLett.98.035703.
- J. Mazur, "Numerical Simulation of Temperature Field in Soil Generated by Solar Radiation," *Journal de Physique IV France* **137** pp. 317-320 (2006), doi:10.1051/jp4:2006137061.
- T. P. Moffat, D. Wheeler, S. K. Kim & D. Josell, "Curvature enhanced adsorbate coverage model for electrodeposition," *Journal of The Electrochemical Society* **153** (2) pp. C127-C132 (2006), 10.1149/1.2165580.
- D. Josell, D. Wheeler & T. P. Moffat, "Gold superfill in submicrometer trenches: Experiment and prediction," *Journal of The Electrochemical Society* **153** (1) pp. C11-C18 (2006), 10.1149/1.2128765.

# FiPy Overview



*The Trilinos Project*



# FiPy Overview

6.2. Module examples.diffusion.mesh20x20

83

to the top-left and bottom-right corners. Neumann boundary conditions are automatically applied to the top-right and bottom-left corners.

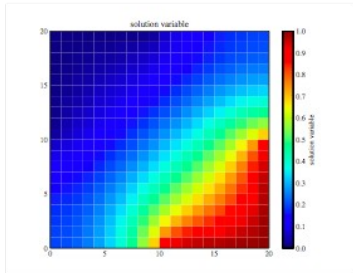
```
>>> x, y = mesh.getFaceCenters()
>>> facesTopLeft = ((mesh.getFacesLeft() & (y > L / 2))
...                 | (mesh.getFacesTop() & (x < L / 2)))
>>> facesBottomRight = ((mesh.getFacesRight() & (y < L / 2))
...                     | (mesh.getFacesBottom() & (x > L / 2)))
>>> BCs = (FixedValue(faces=facesTopLeft, value=valueTopLeft),
...        FixedValue(faces=facesBottomRight, value=valueBottomRight))
```

We create a viewer to see the results

```
>>> if __name__ == '__main__':
...     viewer = Viewer(vars=phi, datamin=0., datamax=1.)
...     viewer.plot()
```

and solve the equation by repeatedly looping in time:

```
>>> timeStepDuration = 10 * 0.9 * dx**2 / (2 * D)
>>> steps = 10
>>> for step in range(steps):
...     eq.solve(vars=phi,
...             boundaryConditions=BCs,
...             dt=timeStepDuration)
...     if __name__ == '__main__':
...         viewer.plot()
```



We can test the value of the bottom-right corner cell.

## 522 high level tests



Build Configuration "trunk" - FiPy - Trac

Repository path: trunk

FiPy trunk. *Should be stable.*

Chgset	Linux	Mac OS X	Mac OS X - fink	Windows	sandbox
[2951]	797: Failed	798: Failed	799: Failed	800: Success	801: Failed
[2950]	szou (71.191.120.120) Darwin 2.6.27-9-generic / i686	paco (1808.58.202) Darwin 8.11.1 / i386	paco (1808.58.202) Darwin 8.11.1 / i386	p611404 (1929.6.153.52) Windows XP	sandbox (129.6.153.79) Linux 2.6.18-6-686 / i686
	0:01:08 checkout	0:00:34 checkout	0:00:22 checkout	0:23:35 checkout	0:00:27 checkout
	0:04:30 test	0:07:09 test	0:08:31 test	0:42:33 test	0:15:33 test
	0:08:48 weave	1 of 476 tests failed	0:15:00 weave	1 of 476 tests failed	0:25:05 weave
	precompile	0:11:01 test	0:09:38 test	0:23:29 test	0:00:08 weave
	0:04:44 test	0:07:53 test	1 of 476 tests failed	0:00:08 test	0:16:13 test
	0:00:01 test	Trilinos failed (256)	0:00:02 test	0:00:06 test	0:00:06 test
	0:00:01 test	0:00:03 test	0:00:01 test	0:00:01 test	0:21:27 test

**Test Coverage**

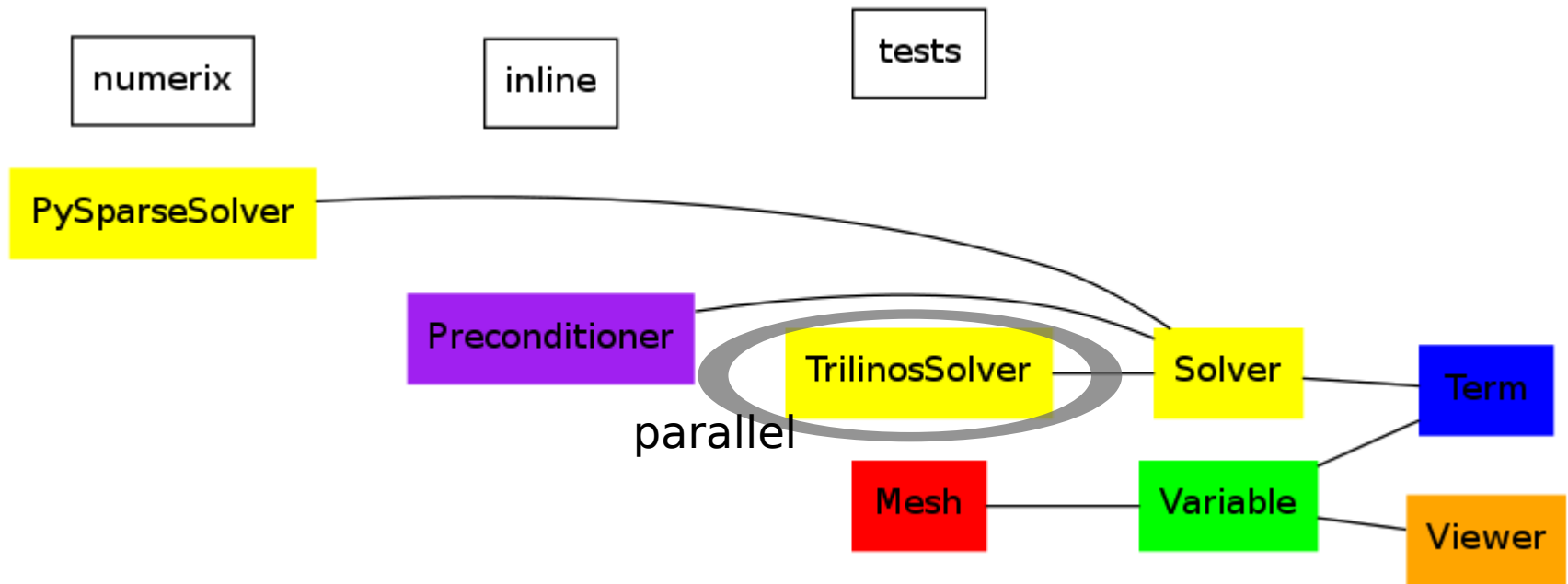
**Unit Tests**



Testing must work transparently in parallel



# FiPy Overview



# Outline

- FiPy example
- FiPy overview
- **Parallel choices**
- Parallel design:
  - Unified communicator Interface
  - Partitioning
  - Testing
- FiPy example in parallel
- Conclusion



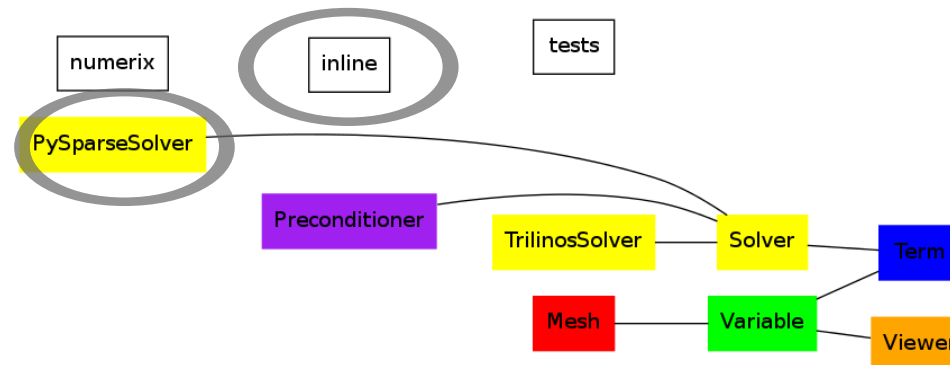
# Parallel Choices

## openmp

```
code = """
#define ITEM(arr,i,vec) (arr[arrayIndex(arr##_array, i, vec)])
int vec[%(rank)d];
#pragma ...
%(loops)s%(indent)s%(code_in)s%(enders)s
#undef ITEM
""" % locals()

weave.inline(code, ...)
```

- Requires changes to low level code
- Many operations are not covered
- Memory isn't parallel
- Too much communication
- Inline requires a code fork (difficult to maintain)

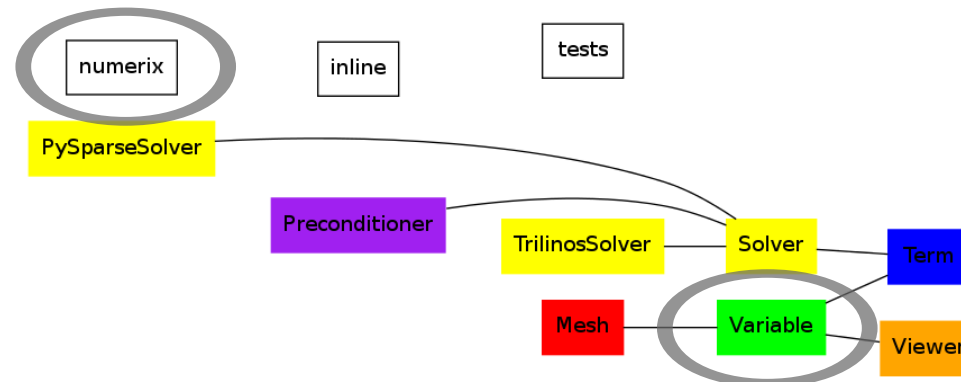


# Parallel Choices

numpy → PyTrilinos.Epetra

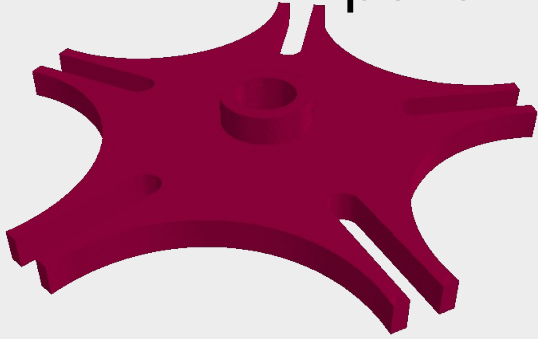
```
class _MeshVariable(Variable):  
    """  
    .. attention:: This class is abstract.  
                   Always create one of its subclasses.  
    """  
    def __init__(self, mesh, name='', value=0., rank=None,  
                 elementsshape=None, unit=None, cached=1):  
  
        if not isinstance(value, Epetra.Vector):  
            value = Epetra.Vector(value, mesh.map)
```

- Pervasive changes
- Epetra is not numpy (fancy indexing etc)
- Not fully parallel
- Trilinos becomes a requirement or code fork



# Parallel Choices

parallel solver

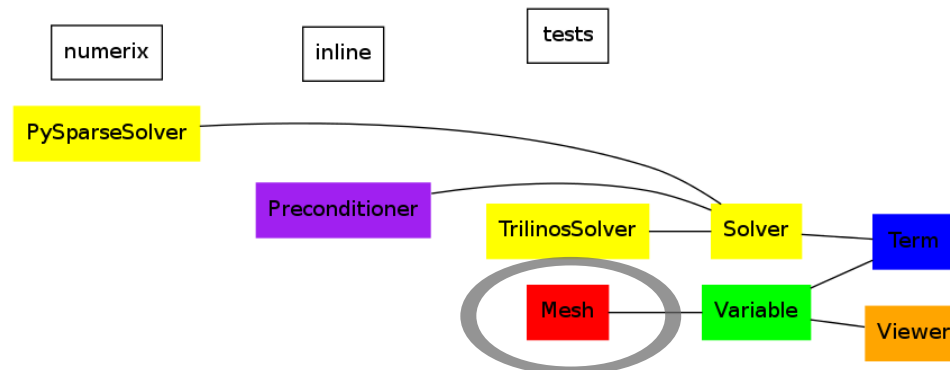


$$\begin{pmatrix} A_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & A_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & A_{2,2} \end{pmatrix}$$

parallel mesh



- Independent FiPy processes with local mesh
- Only the solver communicates
- Only the mesh needs to know about overlaps
- No changes to numerix
- Everything is fully parallel
- Communication is minimal required
- No code forks
- FiPy script is independent of parallel constructs

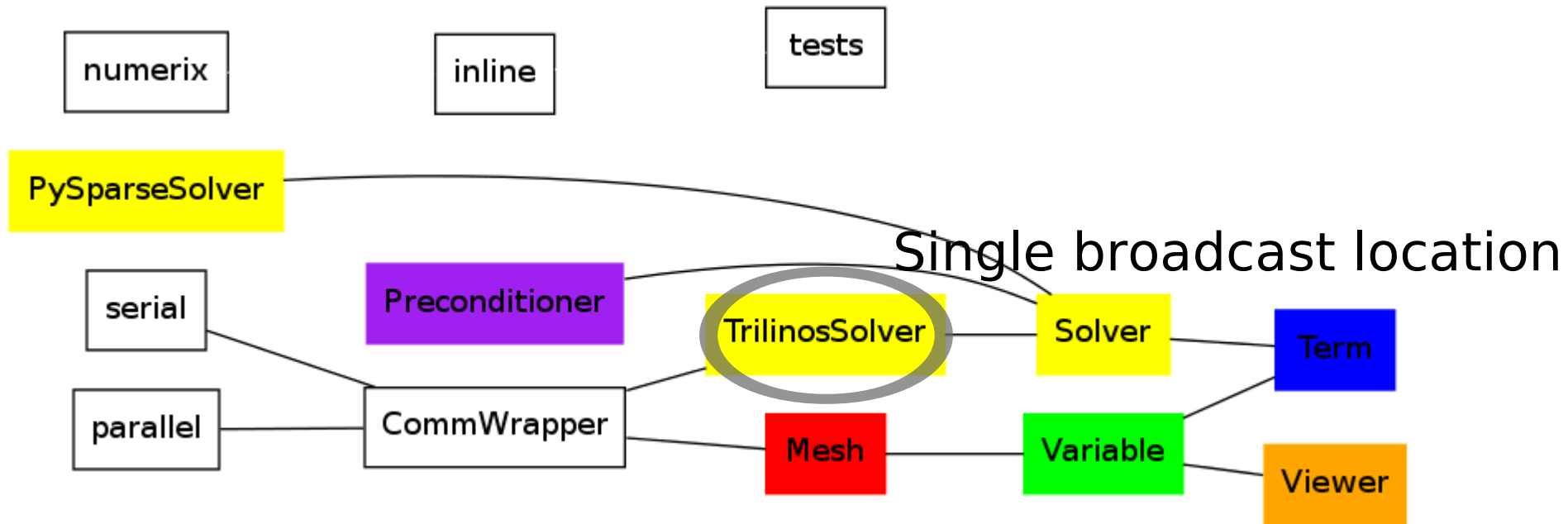


# Outline

- FiPy example
- FiPy overview
- Parallel choices
- **Parallel design:**
  - **Unified communicator Interface**
  - **Partitioning**
  - **Testing**
- FiPy example in parallel
- Conclusion



# Parallel Design



```

from fipy.tools import serial
class Mesh(object):
    """Generic mesh class.
    """
    def __init__(self, vertexCoords, faceVertexIDs, cellFaceIDs, communicator=serial):
        """faceVertexIds and cellFacesIds must be padded with minus ones."""
        self.communicator = communicator

    @property
    def _globalNonOverlappingCellIDs(self):
        return numerix.arange(self.numberOfCells)
    @property
    def _globalOverlappingCellIDs(self):
        return numerix.arange(self.numberOfCells)
    @property
    def _localNonOverlappingCellIDs(self):
        return numerix.arange(self.numberOfCells)
    @property
    def _localOverlappingCellIDs(self):
        return numerix.arange(self.numberOfCells)

```

# Unified Communicator Interface

```
class CommWrapper(object):
    """ Encapsulates capabilities needed for Epetra. Some capabilities are not parallel.
    """
    def __init__(self, Epetra):
        self.epetra_comm = Epetra.PyComm()
class Mpi4pyCommWrapper(CommWrapper):
    """Encapsulates capabilities needed for both Epetra and mpi4py.
    """
    def __init__(self, Epetra, MPI):
        self.MPI = MPI
        self.mpi4py_comm = self.MPI.COMM_WORLD
        CommWrapper.__init__(self, Epetra)
class SerialComm(CommWrapper):
    @property
    def Nproc(self):
        return 1
```

Maintain state in single location

“`from PyTrilinos import ...`” confined to four locations

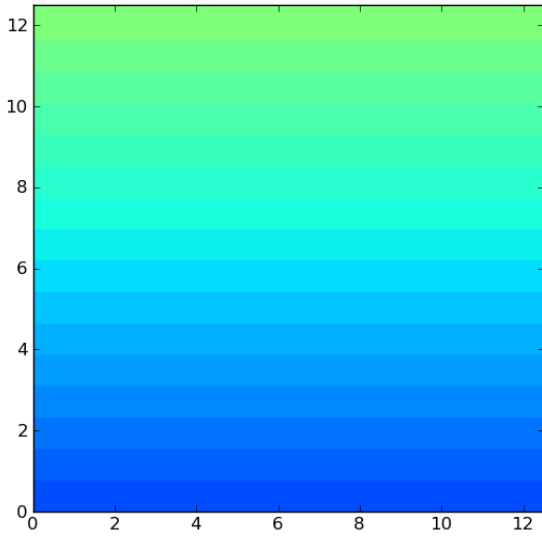
```
./fipy/solvers/trilinos/...
./setup.py
./fipy/matrices/trilinosMatrix.py
./fipy/tools/commWrapper.py
```

```
from fipy import *
from fipy.tools import serial ## serial or parallel
m = Grid2D(nx=2, ny=2, communicator=serial)
```

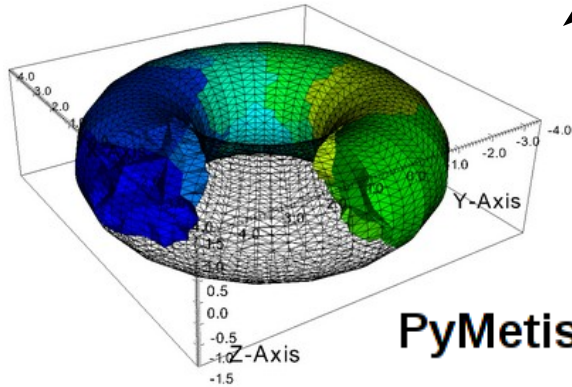
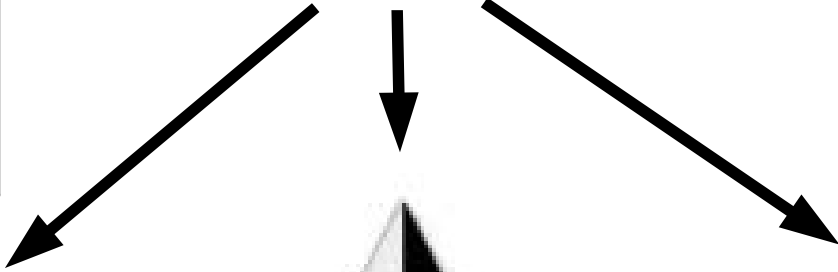
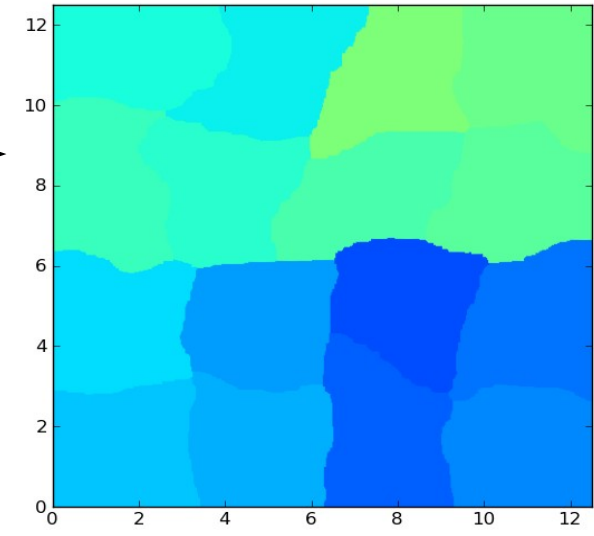
Useful for testing



# Partitioning



**METIS - Serial Graph Partitioning**



**PyMetis**

no overlaps



**Gmsh 2.5**

overlaps



**Zoltan** Parallel Dynamic Load Balancing

not part of PyTrilinos

- FiPy has existing "GmshImport" .msh reader
- "gmshImport" uses `os.system("gmsh ...")`, no direct interface
- requires .geo scripting
- parsing .msh files, difficult in parallel
- only one layer of overlaps
- Gmsh needs a "pythonic" python wrapper



# Parallel Testing

```
$ mpirun -np 2 python setup.py test [--trilinos --pysparse --scipy --pyamg]
running test
running test
...
...
-----
Ran 522 tests in 502.188s

FAILED (failures=3)
```

solver choices

## test.py doctests need to work in parallel

```
"""
>>> from fipy import *
>>> m = Grid1D(nx=1)
>>> v = CellVariable(mesh=m, value=0)
>>> v.constrain(1, m.facesLeft)
>>> DiffusionTerm().solve(v)
>>> print v
[ 1.]
"""
```

```
def _test(self):
    import doctest
    return doctest.testmod()

if __name__ == "__main__":
    _test()
```

many small tests in FiPy

```
$ python
>>> from fipy import *
>>> m = Grid1D(nx=0)
>>> v = CellVariable(mesh=m, value=0)
>>> v.constrain(1, m.facesLeft)
>>> DiffusionTerm().solve(v)
>>> print v
[]
```

```
def __str__(self):
    return str(self.globalValue)
```

```
$ mpirun -np 4 python test.py --trilinos
ok
ok
ok
ok
```



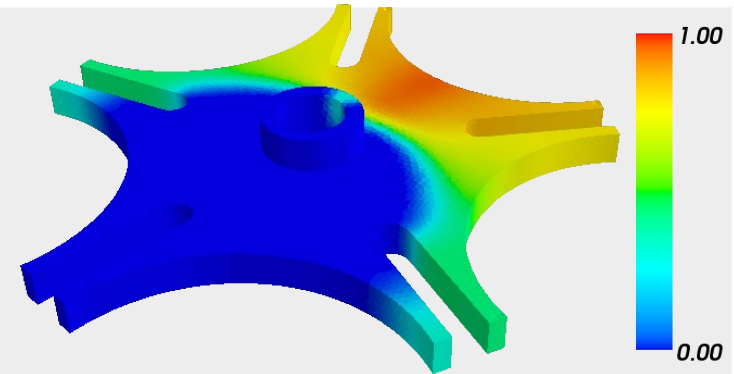
# Outline

- FiPy example
- FiPy overview
- Parallel choices
- Parallel design:
  - Unified communicator Interface
  - Partitioning
  - Testing
- **FiPy example in parallel**
- Conclusion



# FiPy Example in Parallel

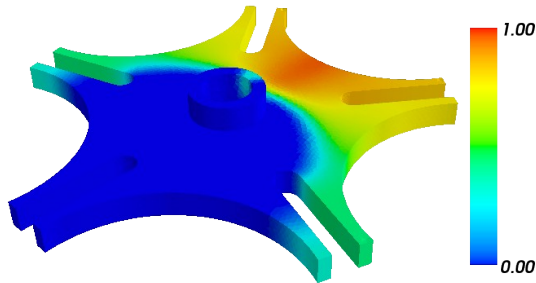
```
## piece.py
from fipy import *
m = Gmsh3D(open('piece.geo').read())
import time
t0 = time.time()
x, y, z = m.cellCenters
phi = CellVariable(mesh=m,
                   value=arccos(x / sqrt(x**2 + y**2)) / pi)
## Viewer(phi).plot()
X, Y, Z = m.faceCenters
phi.faceValue.constrain(value=0,
                        where=(Z == -0.2))
eq = (TransientTerm() == DiffusionTerm(1 / (phi + 1e-3)**2) + 0.1 * phi)
for i in range(5):
    eq.solve(phi, dt=1e-3)
## Viewer(phi).plot()
from fipy.tools import parallel
print 'time:',parallel.procID,time.time() - t0
print 'global cells:',parallel.procID,m.globalNumberOfCells
print 'local cells:',parallel.procID,m.numberOfWorkers
```



```
$ python piece.py
time: 1 85.5958778858
global cells: 0 232887
local cells: 0 232887
```

```
$ mpirun -np 2 python piece.py
time: 1 56.2974278927
time: 0 56.297506094
global cells: 1 232887
local cells: 1 119554
global cells: 0 232887
local cells: 0 119399
```

# FiPy Example in Parallel



8 dual-core AMD Opteron (1024 KB)

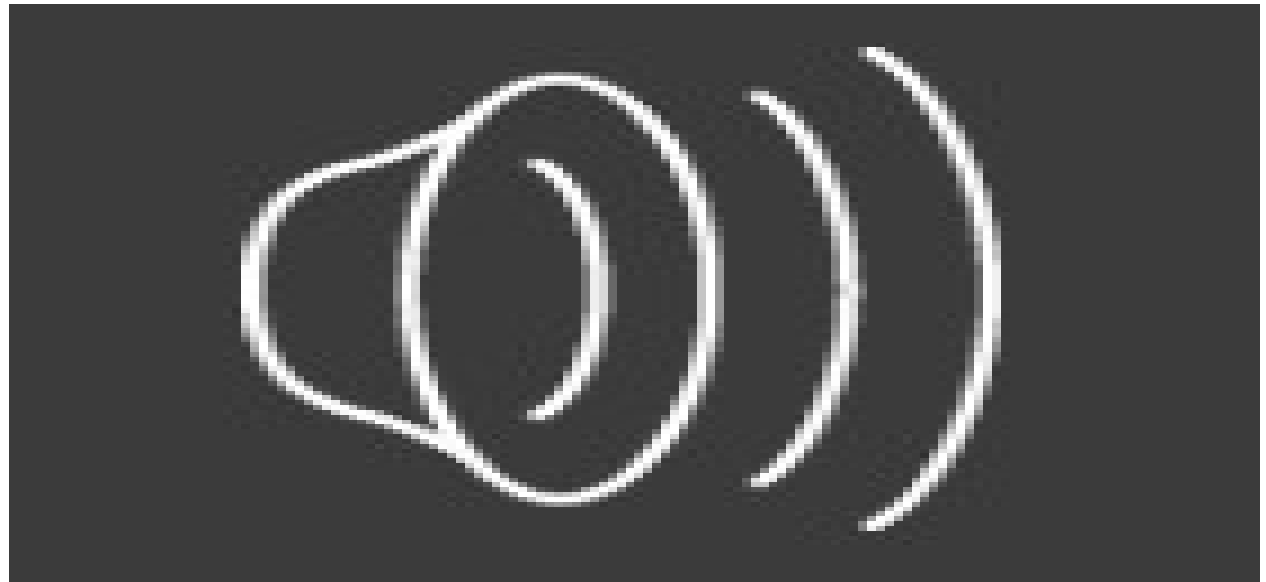
Processors	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)
1	129	129	129
2	59	60	73
4	29	33	29
8	17	17	18

3 × 4 Intel Core 2 Quad CPU (4096 KB)

Processors	Trial 1 (s)	Trial 2 (s)	Trial 3 (s)
1	85	85	86
2	56	49	46
4	39	39	37
8	29	29	28
12	27	27	26

# Reactive Wetting in Parallel

5 fully coupled equations



64 IA-64,  $900 \times 900$

Processors	Sweep 1 (s)	Sweep 2 (s)
1	678	
2	383	304
4	231	193
8	108	91
16	56	52
32	31	29
64	17	14

64 IA-64,  $1800 \times 1800$

Processors	Sweep 1 (s)	Sweep 2 (s)
16	232	188
32	142	111
64	93	70

# Conclusion

