

Modeling of Materials with Python

J. E. Guyer, D. Wheeler & J.A. Warren

www.ctcms.nist.gov/fipy

Metallurgy Division &
Center for Theoretical and Computational Materials Science
Materials Science and Engineering Laboratory

NIST: Origins

“Uniformity in the currency, weights, and measures of the United States is an object of great importance, and will, I am persuaded, be duly attended to.”

George Washington, State of the Union Address, 1790

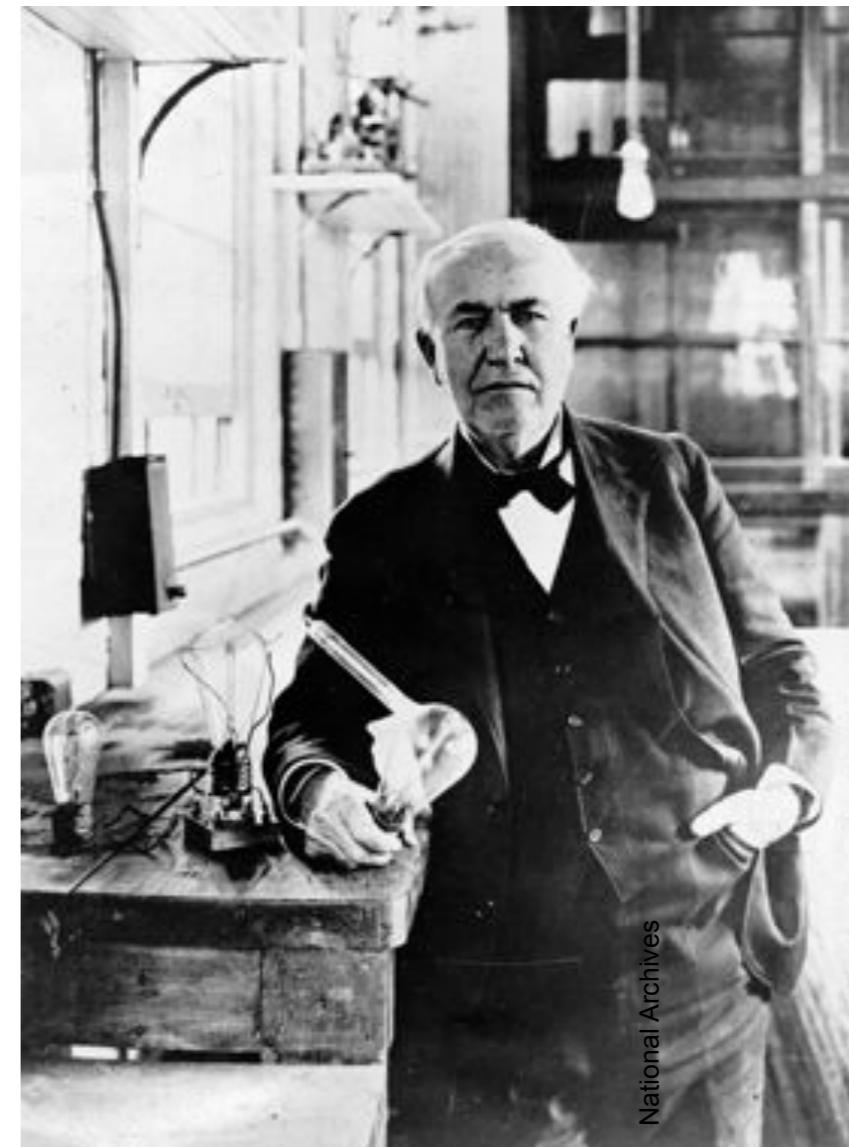


U. S. Constitution, Article I, Section 8

NIST: Origins

National Bureau of Standards
Established by Congress in 1901

- Eight different “authoritative” values for the gallon
- Nascent electrical industry needed standards
- American instruments sent abroad for calibration
- Consumer products and construction materials uneven in quality and unreliable



NIST: Rigorous and Objective



NIST:World-Class Staff



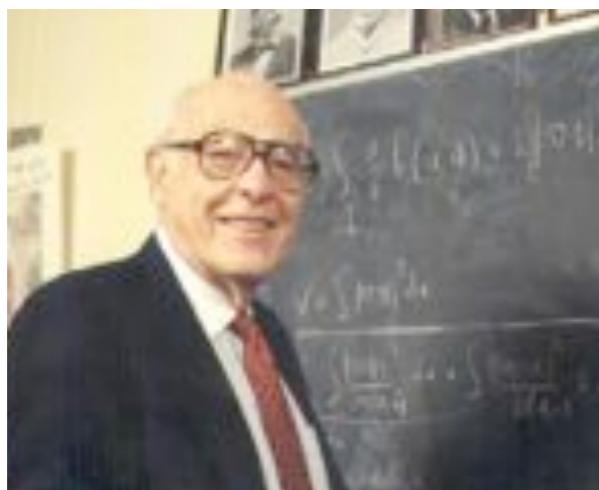
Jan Hall
2005 Nobel Prize
in Physics



Eric Cornell
2001 Nobel Prize
in Physics



Bill Phillips
1997 Nobel Prize
in Physics



John Cahn
1998 National Medal of
Science



Anneke Sengers
2003 L'Oréal-UNESCO
Women in Science Award



Debbie Jin
2003 MacArthur Fellowship

NIST: Trusted Investigator



Great Baltimore Fire, 1904



World Trade Center Collapse
9/11/2001



Silver Bridge Collapse, 1967

NIST: Today

≈2900 employees

≈2600 associates and facilities users

≈1600 field staff in partner organizations

≈400 NIST staff serving on 1000 national and international standards committees

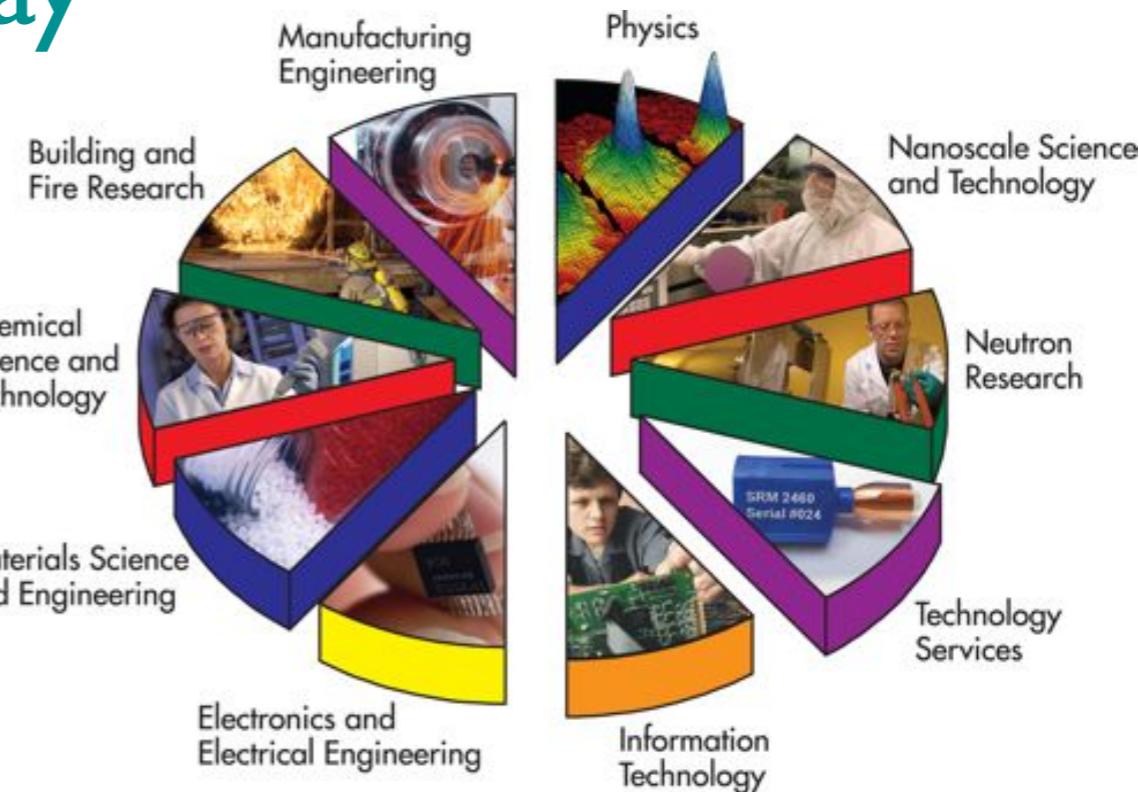
Gaithersburg, MD



Courtesy HDR Architecture, Inc./Steve Hall ©Hedrich Blessing



©Geoffrey Wheeler



Boulder, CO

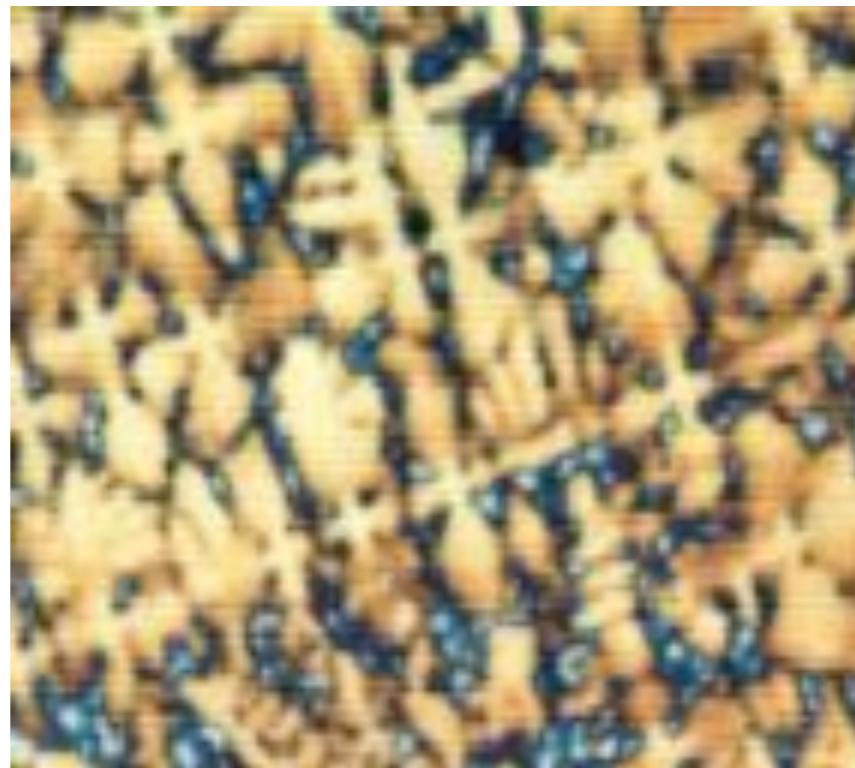


- ≈100 Standard Reference Data sets
- ≈1300 Standard Reference Materials
- ≈16000 calibration tests per year
- ≈800 accreditations per year

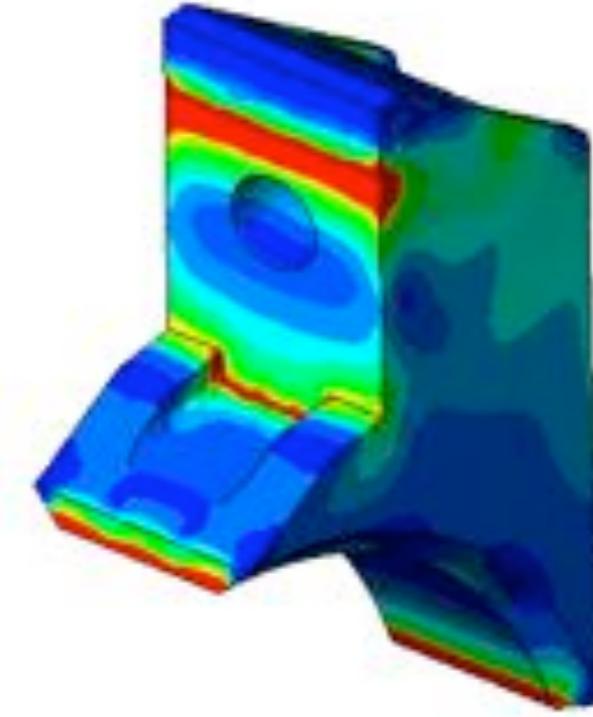
Materials Science



Processing



Structure

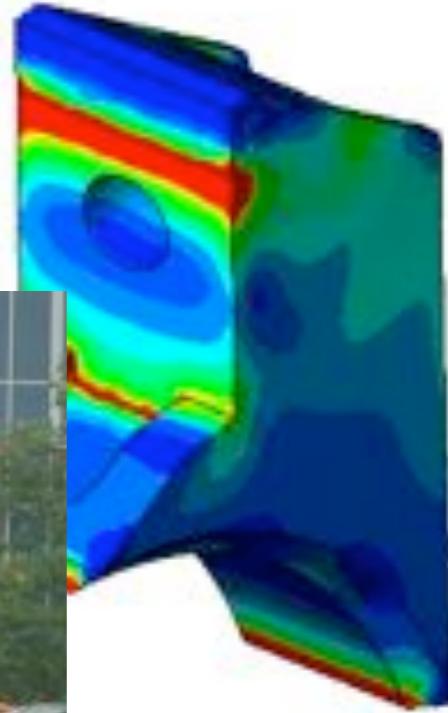


Properties

Materials Science



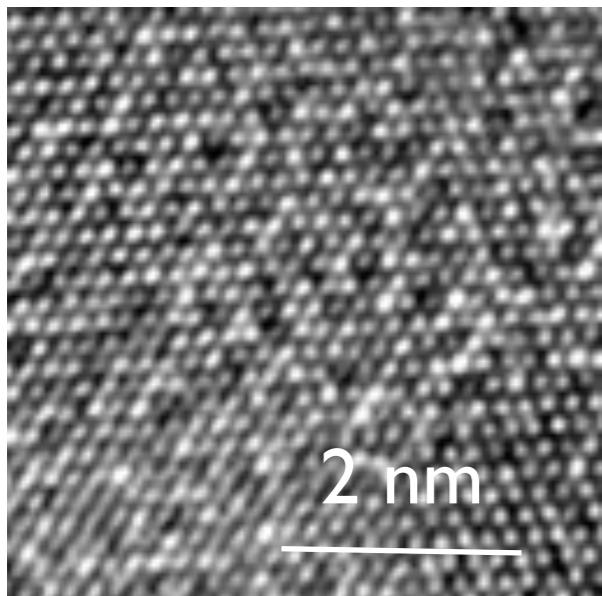
Processing



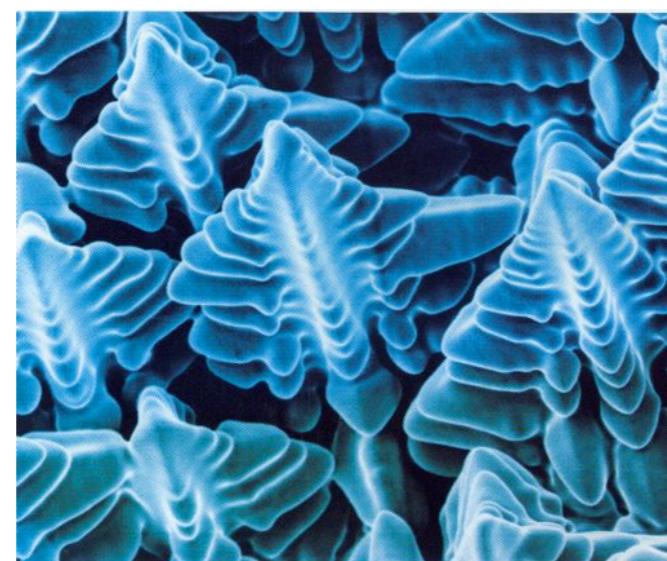
Properties

Performance

Materials And Models



Atomic scale



Micro scale

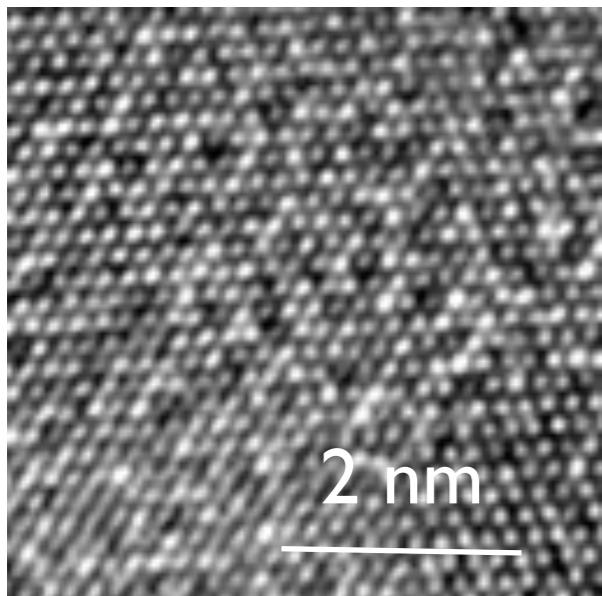


Millimeter scale

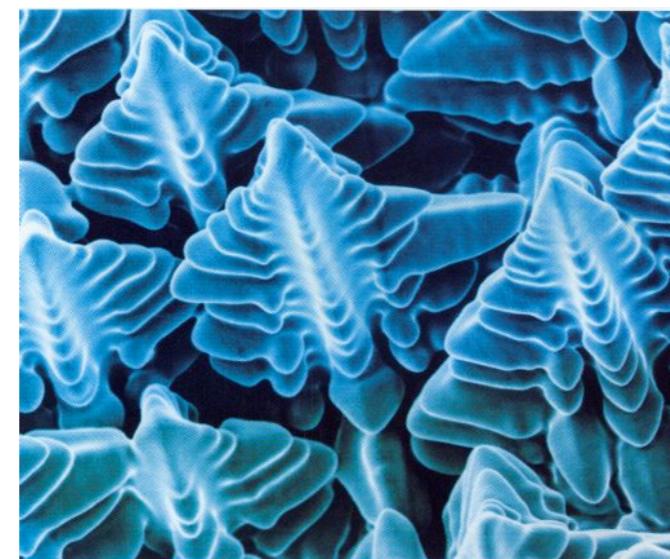


Component scale

Materials And Models



Atomic scale



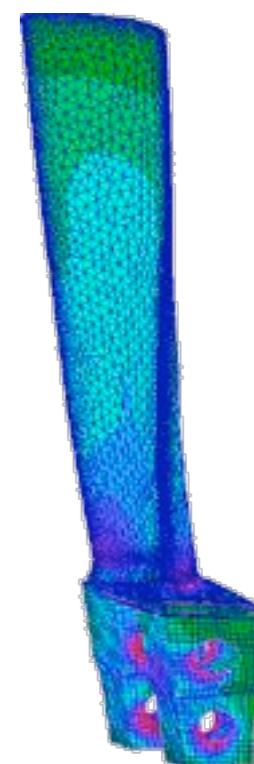
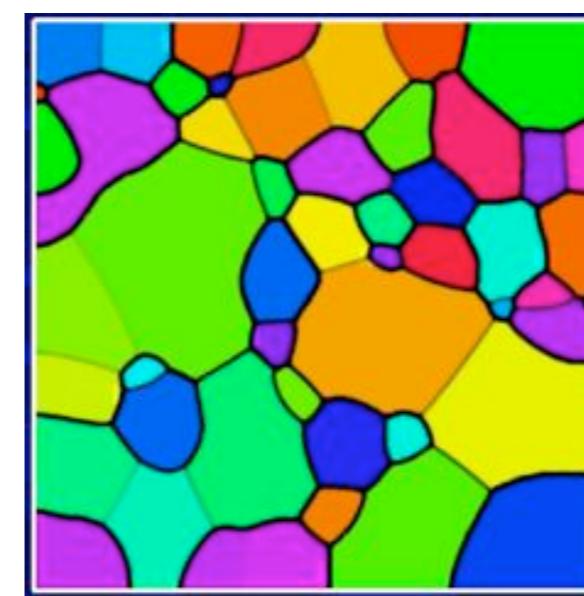
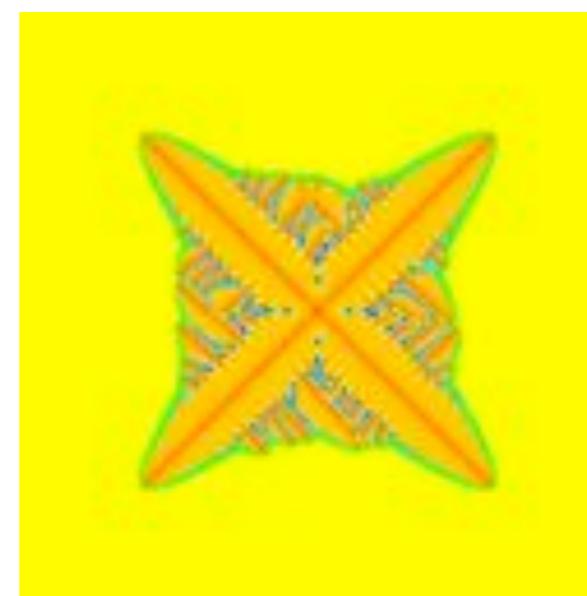
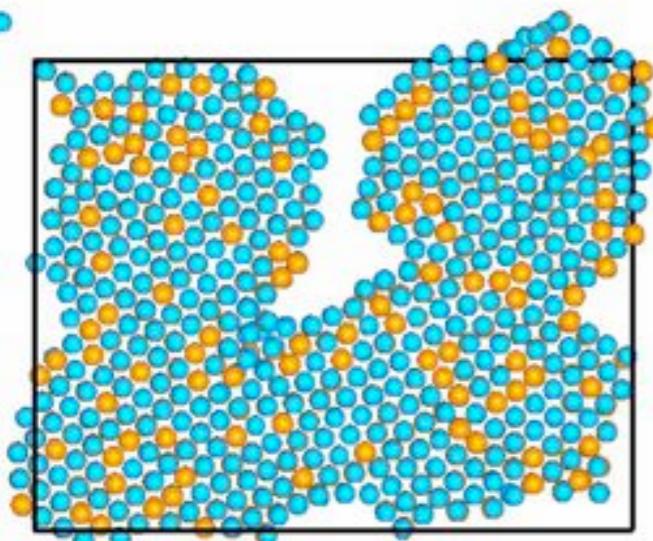
Micro scale



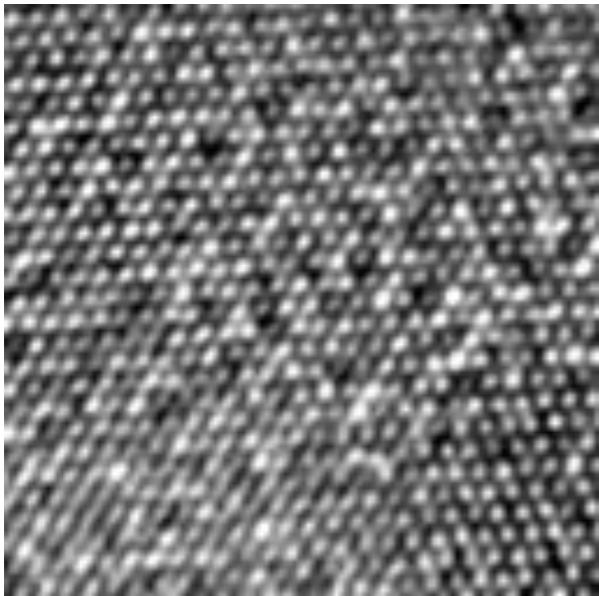
Millimeter scale



Component scale



Modeling is the basis of Measurement Science



Electronic Properties



Heat and Energy

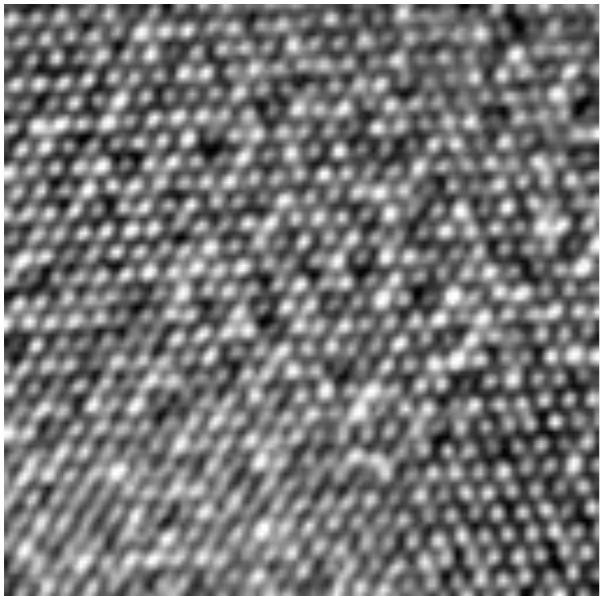


Strength



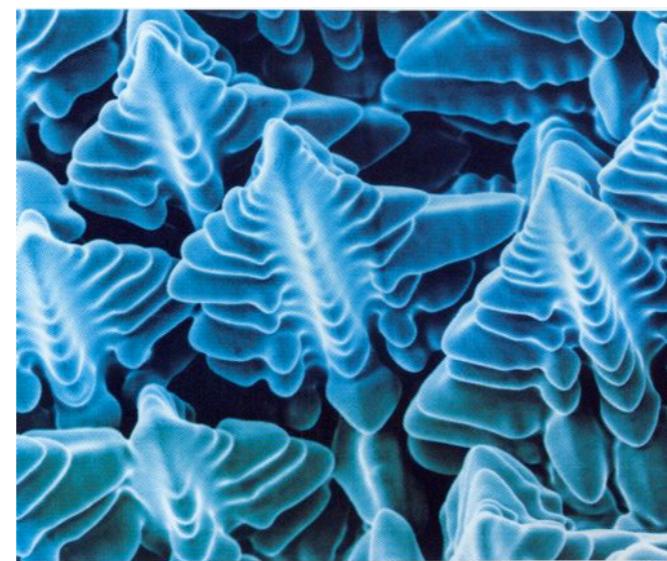
Diffusion

Modeling is the basis of Measurement Science



Electronic Properties

$$V = IR$$



Heat and Energy

$$dE = TdS - PdV + \mu dN$$



Strength

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}$$



Diffusion

$$\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2}$$



Center for Theoretical and Computational Materials Science

Provides the infrastructure for Theory, Modeling and Simulation in MSEL

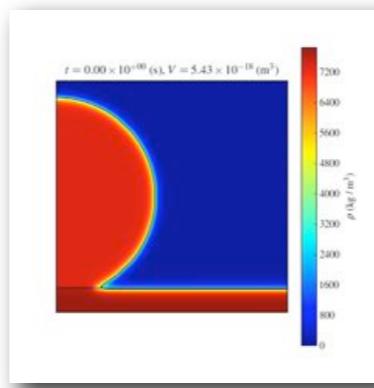
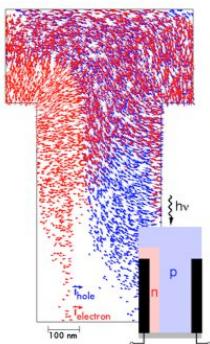
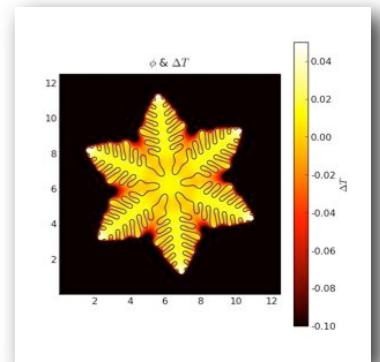
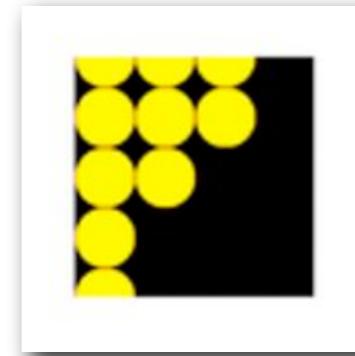
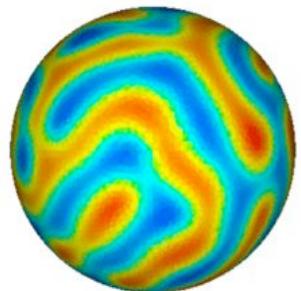
- By providing hardware, and a collaborative work environment (space)
- By developing new measurement paradigms using simulation and theory
- By encouraging the development of software tools, both for internal use and to aid in the dissemination of existing results

Why a common code?

- PDEs are ubiquitous in materials science
- Many codes for solving materials science problems at NIST
- Dissemination to other researchers is “difficult”

Why a common code?

- PDEs are ubiquitous in ~~materials~~ science
- Many codes for solving materials science problems at NIST
- Dissemination to other researchers is “difficult”



Why Scripting?

- hard-code

:

:

common /params/temperature,steps,tolerance,...

data temperature,steps,tolerance/273.15,10000,1.d-3,...

:

:

Why Scripting?

- hard-code

:
:

common /params/temperature,steps,tolerance,...

data temperature,steps,tolerance/273.15,10000,1.d-3,...

:

:

- batch file

273.15

10000

1.d-3

:

:

Why Scripting?

- hard-code

```
:  
:  
common /params/temperature,steps,tolerance,...
```

```
data temperature,steps,tolerance/273.15,10000,1.d-3,...
```

```
:
```

- batch file

```
273.15  
10000  
1.d-3  
:  
:
```

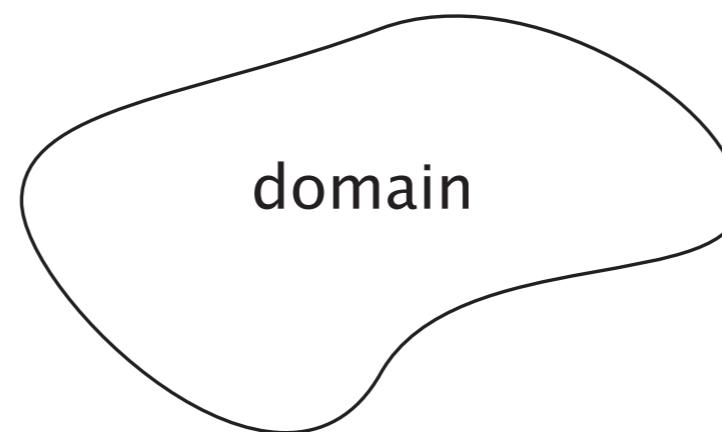
- comment character

```
273.15 # temperature (K)  
10000 # steps  
1.d-3 # tolerance  
:  
:
```

Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ

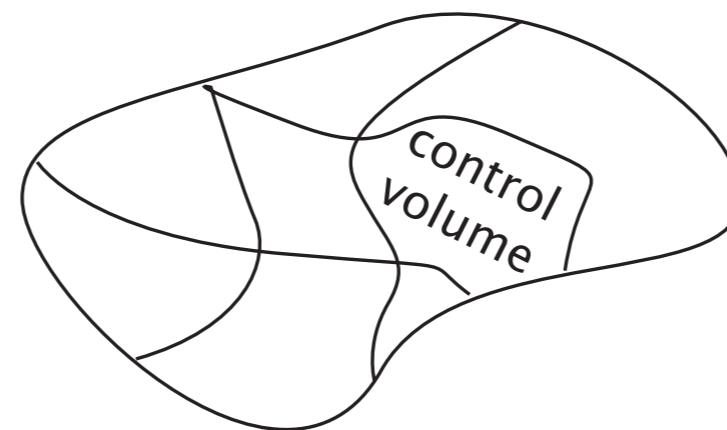
$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{transient}} - \underbrace{[\nabla \cdot (\Gamma_i \nabla)]^n \phi}_{\text{diffusion}} - \underbrace{\nabla \cdot (\vec{u}\phi)}_{\text{convection}} - \underbrace{S_\phi}_{\text{source}} = 0$$



Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ
- Integrate PDE over arbitrary control volumes

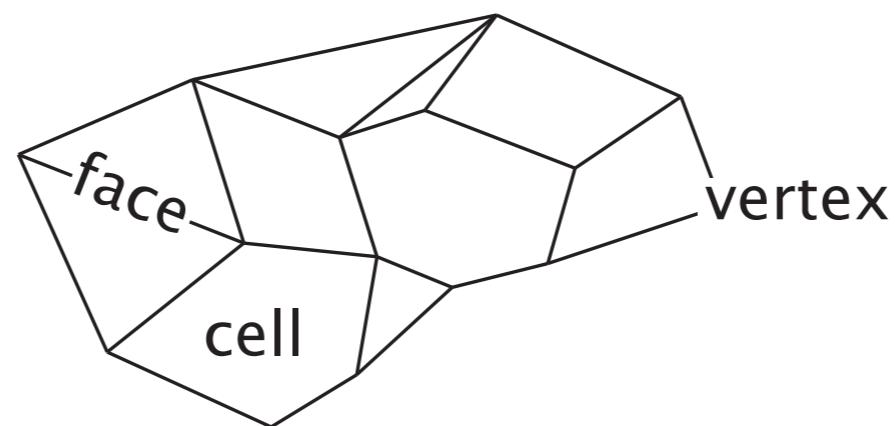
$$\underbrace{\int_V \frac{\partial(\rho\phi)}{\partial t} dV}_{\text{transient}} - \underbrace{\int_S \Gamma_n (\vec{n} \cdot \nabla \{\dots\}) dS}_{\text{diffusion}} - \underbrace{\int_S (\vec{n} \cdot \vec{u})\phi dS}_{\text{convection}} - \underbrace{\int_V S_\phi dV}_{\text{source}} = 0$$



Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ
- Integrate PDE over arbitrary control volumes
- Evaluate PDE over polyhedral control volumes

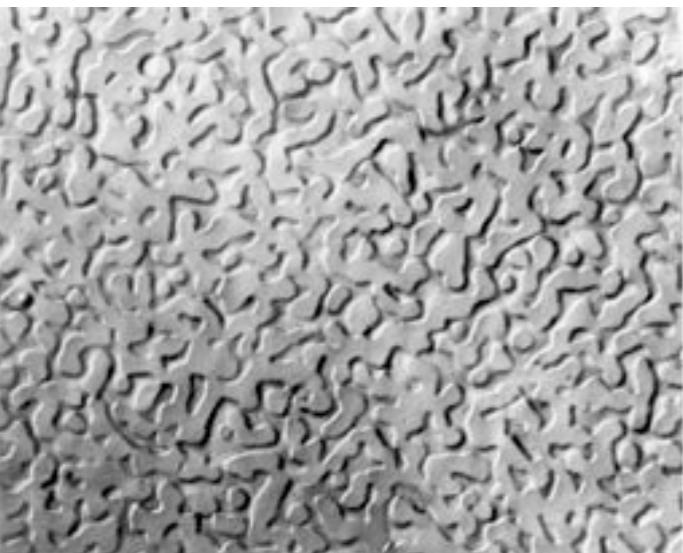
$$\underbrace{\frac{\rho\phi V - (\rho\phi V)^{\text{old}}}{\Delta t}}_{\text{transient}} - \underbrace{\sum_{\text{face}} [\Gamma_n A \vec{n} \cdot \nabla \{\dots\}]_{\text{face}}}_{\text{diffusion}} - \underbrace{\sum_{\text{face}} [(\vec{n} \cdot \vec{u}) A \phi]_{\text{face}}}_{\text{convection}} - VS_\phi = 0$$



Finite Volume Method

- Solve a general PDE on a given domain for a field ϕ
- Integrate PDE over arbitrary control volumes
- Evaluate PDE over polyhedral control volumes
- Obtain a large coupled set of linear equations in ϕ

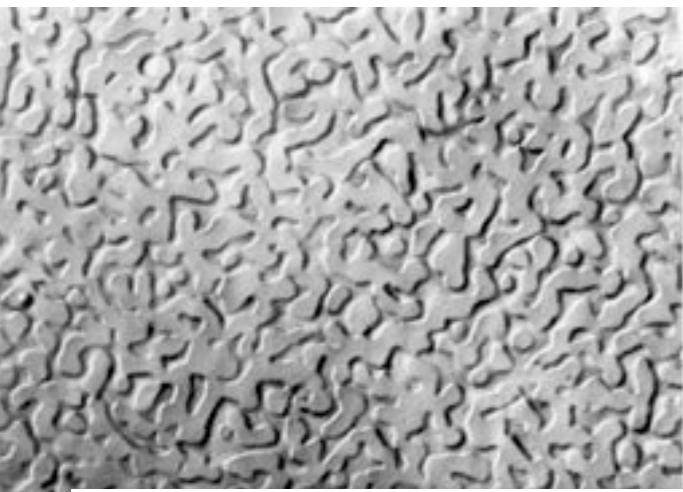
$$\begin{pmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & a_{nn} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$



Cahn-Hilliard Example

$$\frac{\partial \phi}{\partial t} = \nabla \cdot D \nabla \left(\frac{\partial f}{\partial \phi} - \epsilon^2 \nabla^2 \phi \right)$$

$$f = \frac{a^2}{2} \phi^2 (1 - \phi)^2$$



Cahn-Hilliard Example

$$\frac{\partial \phi}{\partial t} = \underbrace{\nabla \cdot D a^2 [1 - 6\phi(1 - \phi)] \nabla \phi}_{\text{transient}} - \underbrace{\nabla \cdot D \nabla \epsilon^2 \nabla^2 \phi}_{\text{2nd order diffusion}} - \underbrace{D \nabla^2 \phi}_{\text{4th order diffusion}}$$

```
from fipy import *
mesh = Grid2D(nx=1000, ny=1000, dx=0.25, dy=0.25)

phi = CellVariable(name=r"\phi", mesh=mesh)
phi.setValue(GaussianNoiseVariable(mesh=mesh,
                                    mean=0.5,
                                    variance=0.01))

PHI = phi.getArithmeticFaceValue()
D = a = epsilon = 1.
eq = (TransientTerm()
      == DiffusionTerm(coeff=D * a**2 * (1 - 6 * PHI * (1 - PHI)))
      - DiffusionTerm(coeff=(D, epsilon**2)))

viewer = Viewer(vars=(phi,), datamin=0., datamax=1.)

dexp = -5
elapsed = 0.
while elapsed < 1000.:
    dt = min(100, exp(dexp))
    elapsed += dt
    dexp += 0.01
    eq.solve(phi, dt=dt)
    viewer.plot()
```

Cahn-Hilliard Example

$$\frac{\partial \phi}{\partial t} = \underbrace{\nabla \cdot D a^2 [1 - 6\phi(1 - \phi)] \nabla \phi}_{\text{transient}} - \underbrace{\nabla \cdot D \nabla \epsilon^2 \nabla^2 \phi}_{\text{2nd order diffusion}} - \underbrace{D \nabla^2 \phi}_{\text{4th order diffusion}}$$

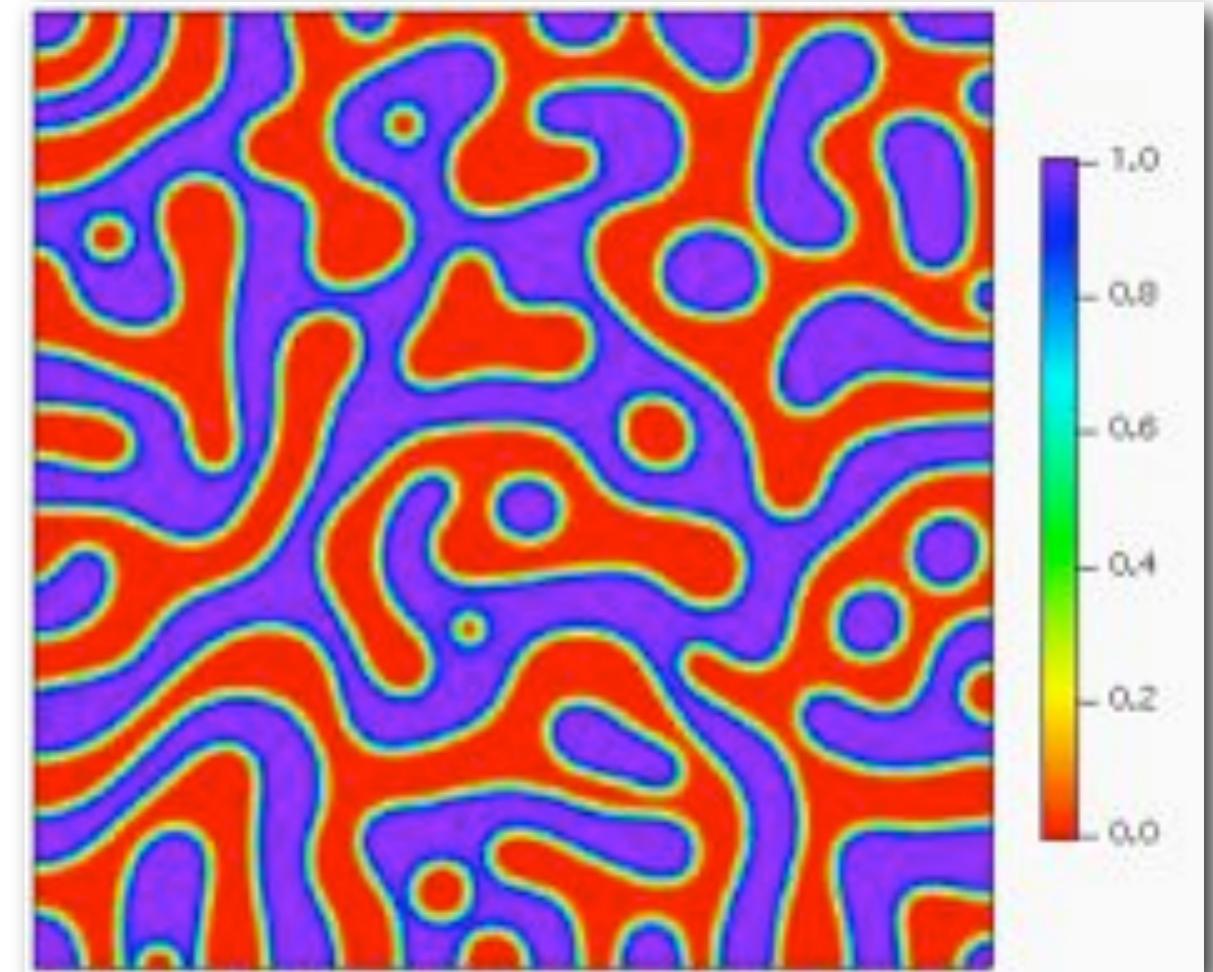
```
from fipy import *
mesh = Grid2D(nx=1000, ny=1000, dx=0.25, dy=0.25)

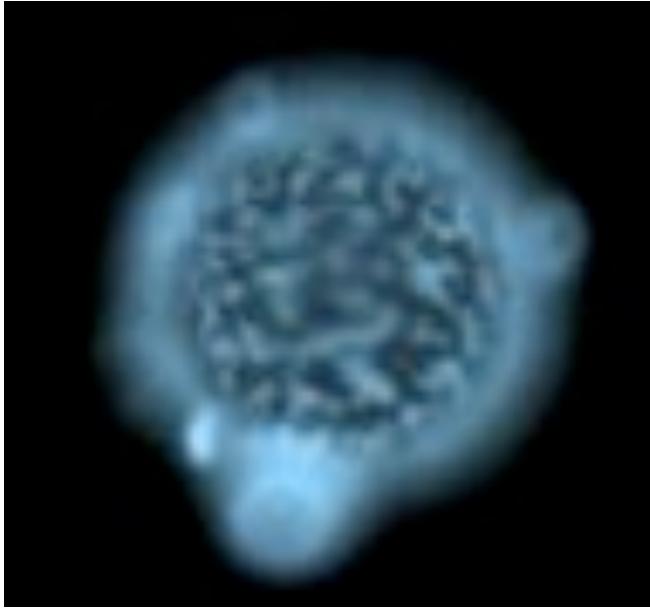
phi = CellVariable(name=r"\phi", mesh=mesh)
phi.setValue(GaussianNoiseVariable(mesh=mesh,
                                    mean=0.5,
                                    variance=0.01))

PHI = phi.getArithmeticFaceValue()
D = a = epsilon = 1.
eq = (TransientTerm()
      == DiffusionTerm(coeff=D * a**2 * (1 - 6 * PHI * (1 - PHI)))
      - DiffusionTerm(coeff=(D, epsilon**2)))

viewer = Viewer(vars=(phi,), datamin=0., datamax=1.)

dexp = -5
elapsed = 0.
while elapsed < 1000.:
    dt = min(100, exp(dexp))
    elapsed += dt
    dexp += 0.01
    eq.solve(phi, dt=dt)
    viewer.plot()
```

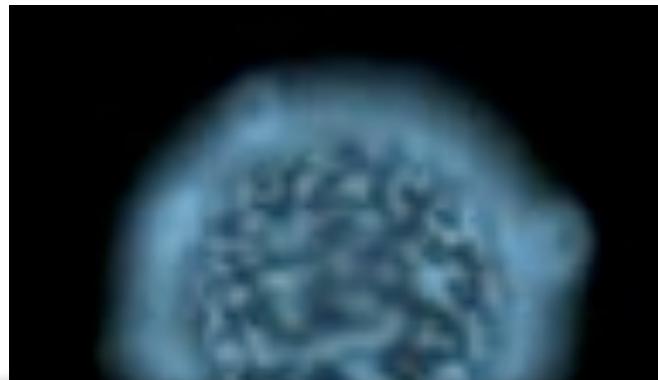




Cahn-Hilliard Example

$$\underbrace{\frac{\partial \phi}{\partial t}}_{\text{transient}} = \underbrace{\nabla \cdot D a^2 [1 - 6\phi(1 - \phi)] \nabla \phi}_{\text{2}^{\text{nd}} \text{ order diffusion}} - \underbrace{\nabla \cdot D \nabla \epsilon^2 \nabla^2 \phi}_{\text{4}^{\text{th}} \text{ order diffusion}}$$

Cahn-Hilliard Example



$$\frac{\partial \phi}{\partial t} = \underbrace{\nabla \cdot D a^2 [1 - 6\phi(1 - \phi)] \nabla \phi}_{\text{transient}} - \underbrace{\nabla \cdot D \nabla \epsilon^2 \nabla^2 \phi}_{\text{2nd order diffusion}} - \underbrace{4^{\text{th}} \text{ order diffusion}}$$

```
mesh = Grid2D(nx 1000, ny 1000, dx 0.25, dy 0.25)
```

```
mesh = GmshImporter2DIn3DSpace("")
```

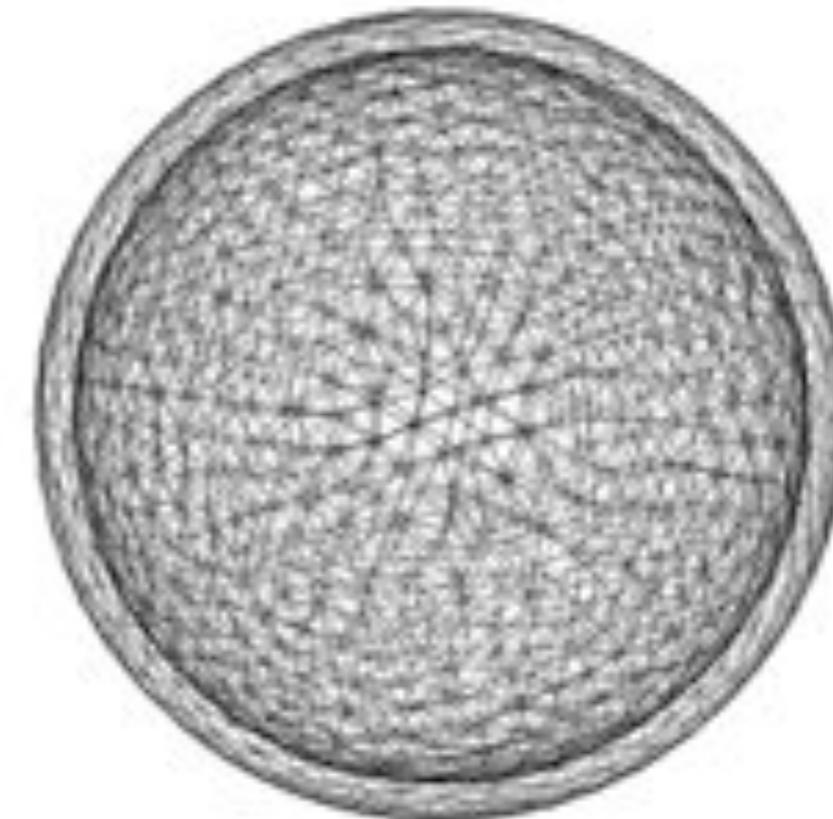
```
radius = 5.0;
```

```
cellSize = 0.3;
```

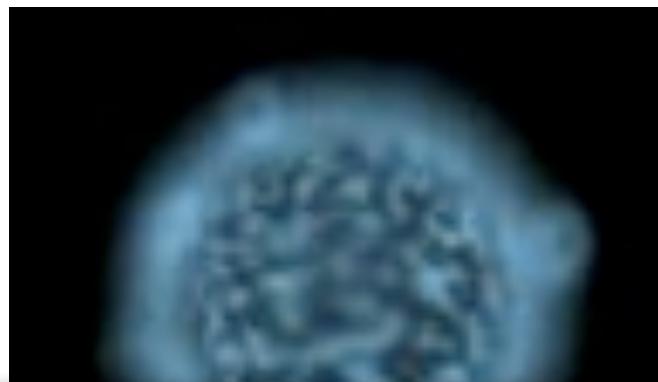
```
// create inner 1/8 shell
Point(1) = {0, 0, 0, cellSize};
Point(2) = {-radius, 0, 0, cellSize};
Point(3) = {0, radius, 0, cellSize};
Point(4) = {0, 0, radius, cellSize};
Circle(1) = {2, 1, 3};
Circle(2) = {4, 1, 2};
Circle(3) = {4, 1, 3};
Line Loop(1) = {1, -3, 2} ;
Ruled Surface(1) = {1};
```

```
// create remaining 7/8 inner shells
t1[] = Rotate {{0,0,1},{0,0,0},Pi/2} {Duplicata{Surface{1}};};
t2[] = Rotate {{0,0,1},{0,0,0},Pi} {Duplicata{Surface{1}};};
t3[] = Rotate {{0,0,1},{0,0,0},Pi*3/2} {Duplicata{Surface{1}};};
t4[] = Rotate {{0,1,0},{0,0,0},-Pi/2} {Duplicata{Surface{1}};};
t5[] = Rotate {{0,0,1},{0,0,0},Pi/2} {Duplicata{Surface{t4[0]}};};
t6[] = Rotate {{0,0,1},{0,0,0},Pi} {Duplicata{Surface{t4[0]}};};
t7[] = Rotate {{0,0,1},{0,0,0},Pi*3/2} {Duplicata{Surface{t4[0]}};};
```

```
// create entire inner and outer shell
Surface Loop(100)={1,t1[0],t2[0],t3[0],t7[0],t4[0],t5[0],t6[0]};
""").extrude(extrudeFunc=lambda r: 1.1 * r)
```



Cahn-Hilliard Example



$$\frac{\partial \phi}{\partial t} = \underbrace{\nabla \cdot D a^2 [1 - 6\phi(1 - \phi)] \nabla \phi}_{\text{transient}} - \underbrace{\nabla \cdot D \nabla \epsilon^2 \nabla^2 \phi}_{\text{2nd order diffusion}} - \underbrace{4^{\text{th}} \text{ order diffusion}}$$

```
mesh = Grid2D(nx 1000, ny 1000, dx 0.25, dy 0.25)
```

```
mesh = GmshImporter2DIn3DSpace("")
```

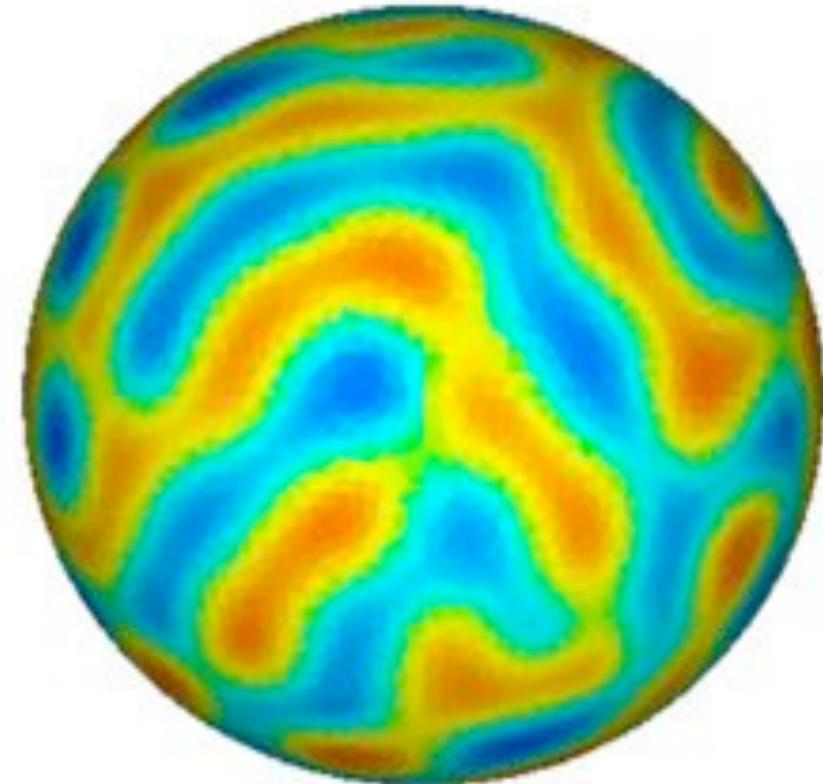
```
radius = 5.0;
```

```
cellSize = 0.3;
```

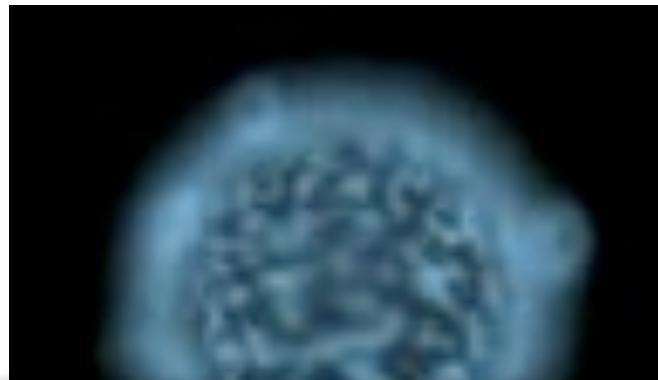
```
// create inner 1/8 shell
Point(1) = {0, 0, 0, cellSize};
Point(2) = {-radius, 0, 0, cellSize};
Point(3) = {0, radius, 0, cellSize};
Point(4) = {0, 0, radius, cellSize};
Circle(1) = {2, 1, 3};
Circle(2) = {4, 1, 2};
Circle(3) = {4, 1, 3};
Line Loop(1) = {1, -3, 2} ;
Ruled Surface(1) = {1};
```

```
// create remaining 7/8 inner shells
t1[] = Rotate {{0,0,1},{0,0,0},Pi/2} {Duplicata{Surface{1}};};
t2[] = Rotate {{0,0,1},{0,0,0},Pi} {Duplicata{Surface{1}};};
t3[] = Rotate {{0,0,1},{0,0,0},Pi*3/2} {Duplicata{Surface{1}};};
t4[] = Rotate {{0,1,0},{0,0,0},-Pi/2} {Duplicata{Surface{1}};};
t5[] = Rotate {{0,0,1},{0,0,0},Pi/2} {Duplicata{Surface{t4[0]}};};
t6[] = Rotate {{0,0,1},{0,0,0},Pi} {Duplicata{Surface{t4[0]}};};
t7[] = Rotate {{0,0,1},{0,0,0},Pi*3/2} {Duplicata{Surface{t4[0]}};};
```

```
// create entire inner and outer shell
Surface Loop(100)={1,t1[0],t2[0],t3[0],t7[0],t4[0],t5[0],t6[0]};
""").extrude(extrudeFunc=lambda r: 1.1 * r)
```



Cahn-Hilliard Example



$$\frac{\partial \phi}{\partial t} = \underbrace{\nabla \cdot D a^2 [1 - 6\phi(1-\phi)] \nabla \phi}_{\text{transient}} - \underbrace{\nabla \cdot D \nabla \epsilon^2 \nabla^2 \phi}_{\text{4th order diffusion}}$$

2nd order diffusion

```
mesh = Grid2D(nx 1000, ny 1000, dx 0.25, dy 0.25)
```

```
mesh = GmshImporter2DIn3DSpace("")
```

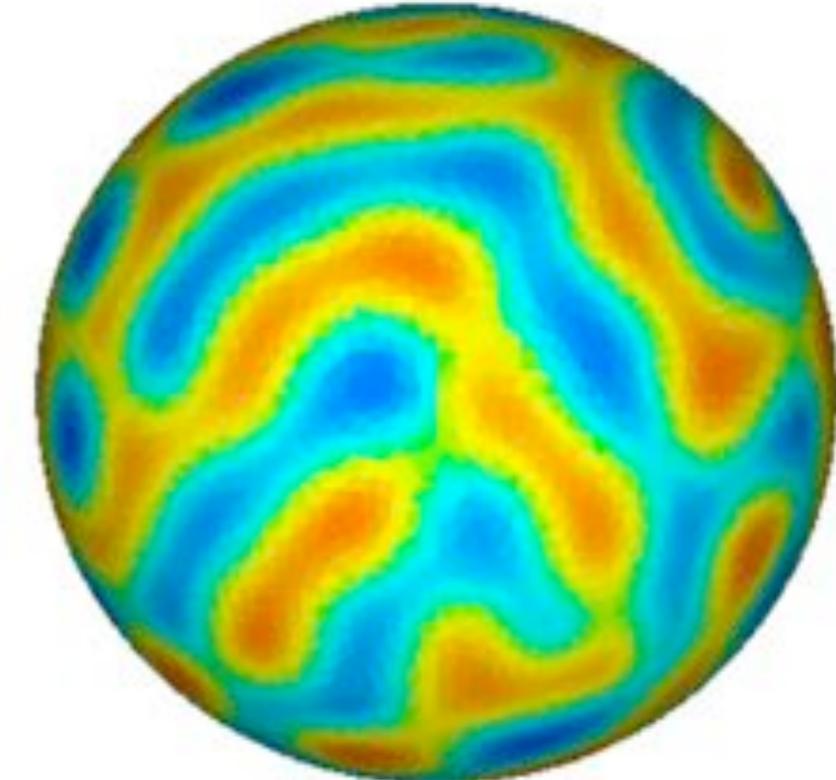
```
radius = 5.0;
```

```
cellSize = 0.3;
```

```
// create inner 1/8 shell
Point(1) = {0, 0, 0, cellSize};
Point(2) = {-radius, 0, 0, cellSize};
Point(3) = {0, radius, 0, cellSize};
Point(4) = {0, 0, radius, cellSize};
Circle(1) = {2, 1, 3};
Circle(2) = {4, 1, 2};
Circle(3) = {4, 1, 3};
Line Loop(1) = {1, -3, 2} ;
Ruled Surface(1) = {1};
```

```
// create remaining 7/8 inner shells
t1[] = Rotate {{0,0,1},{0,0,0},Pi/2} {Duplicata{Surface{1}};};
t2[] = Rotate {{0,0,1},{0,0,0},Pi} {Duplicata{Surface{1}};};
t3[] = Rotate {{0,0,1},{0,0,0},Pi*3/2} {Duplicata{Surface{1}};};
t4[] = Rotate {{0,1,0},{0,0,0},-Pi/2} {Duplicata{Surface{1}};};
t5[] = Rotate {{0,0,1},{0,0,0},Pi/2} {Duplicata{Surface{t4[0]}};};
t6[] = Rotate {{0,0,1},{0,0,0},Pi} {Duplicata{Surface{t4[0]}};};
t7[] = Rotate {{0,0,1},{0,0,0},Pi*3/2} {Duplicata{Surface{t4[0]}};};
```

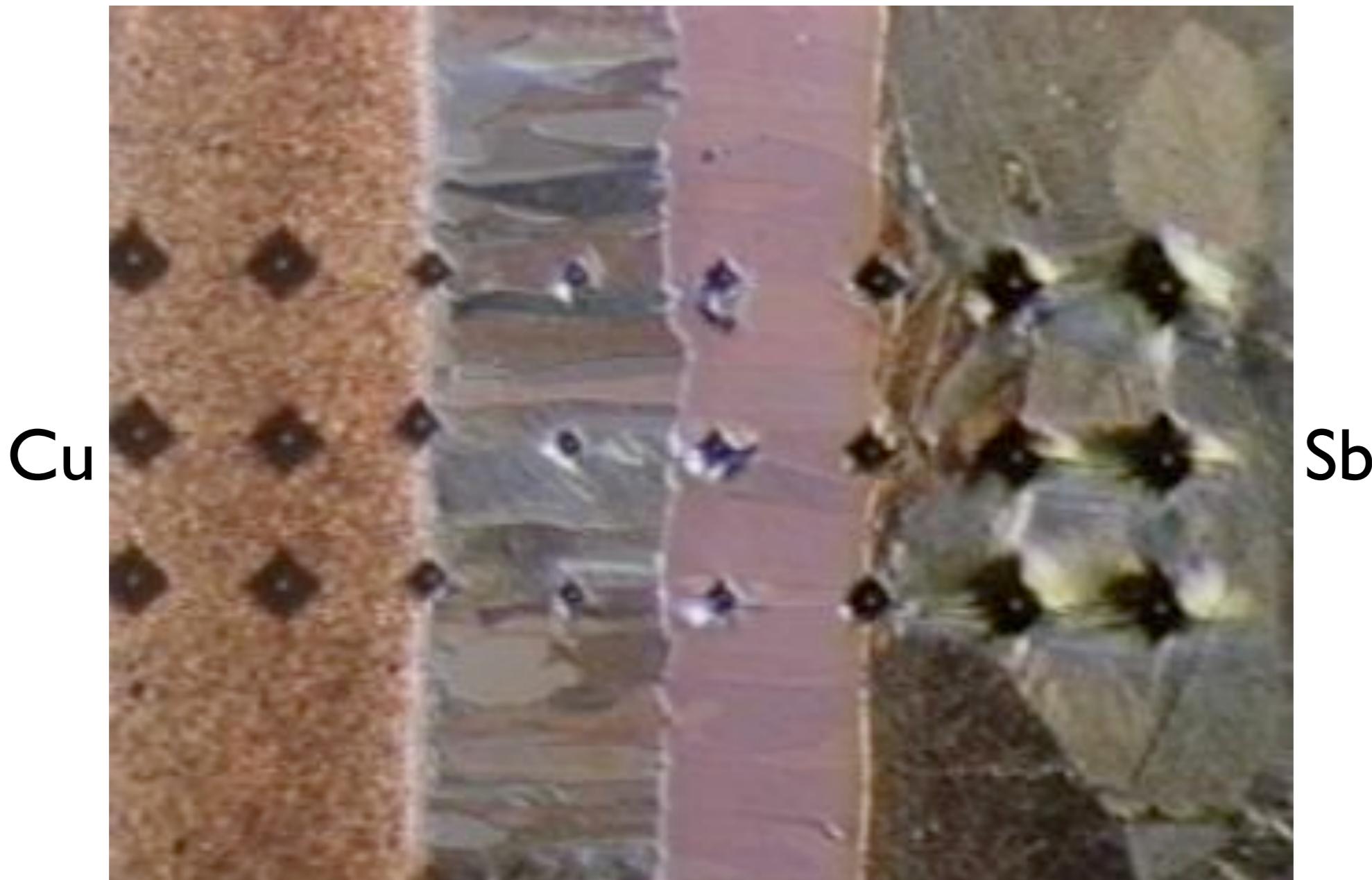
```
// create entire inner and outer shell
Surface Loop(100)={1,t1[0],t2[0],t3[0],t7[0],t4[0],t5[0],t6[0]};
""").extrude(extrudeFunc=lambda r: 1.1 * r)
```



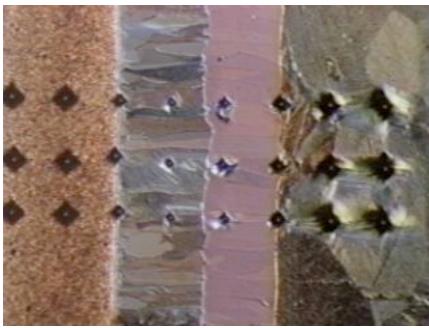
compare, e.g.,

youtube.com/watch?v=kDsFP67_ZSE

Kirkendall effect



Heumann & Ruth © IWF, Göttingen 1986



Kirkendall effect

W. J. Boettinger, J. E. Guyer,
C. E. Campbell & G. B. McFadden,
Proceedings of the Royal Society A: Mathematical
463 (2007) 3347–3373

Cahn-Hilliard treatment

$$\frac{\partial X_B}{\partial t} = \frac{\partial}{\partial z} \left\{ \tilde{M}(X_B) \frac{\partial}{\partial z} \left[-\Omega(X_B - X_A) + RT(\ln X_B - \ln X_A) - 2K \frac{\partial^2 X_B}{\partial z^2} \right] \right\}$$

$$\tilde{M} = X_A X_B [X_B M_A + X_A M_B]$$

$$M_A = M_0 [\beta_1(1 - X_B) + \beta_2 X_B]$$

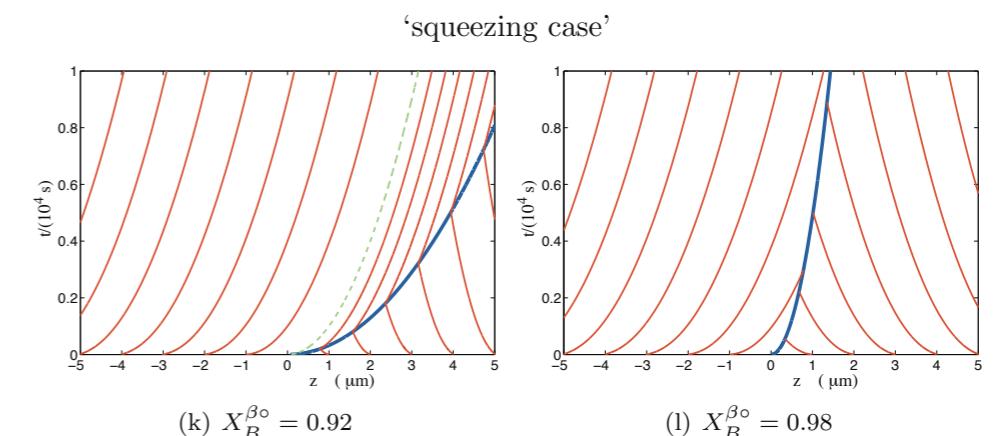
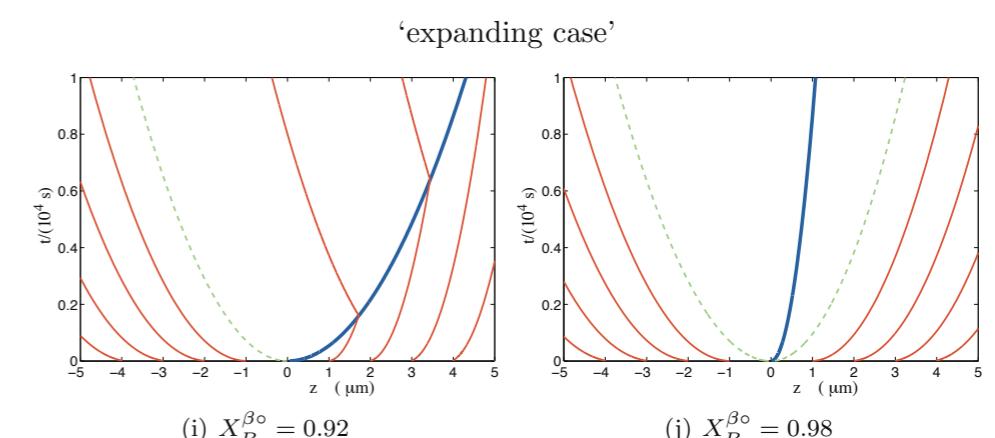
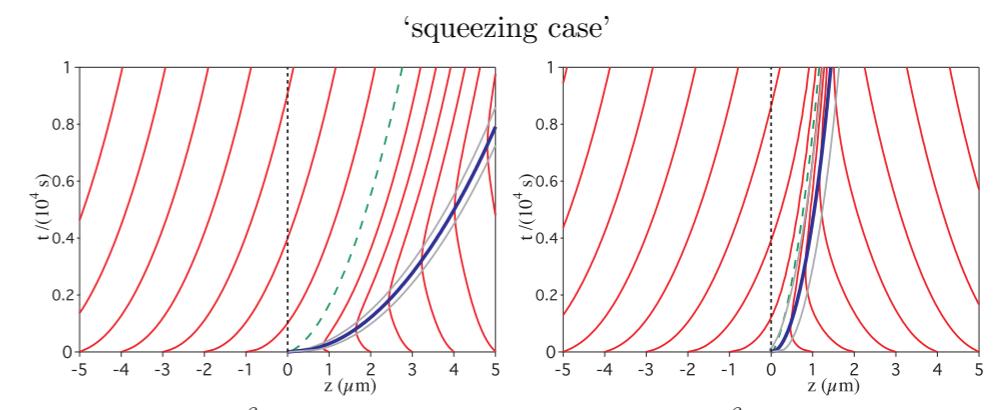
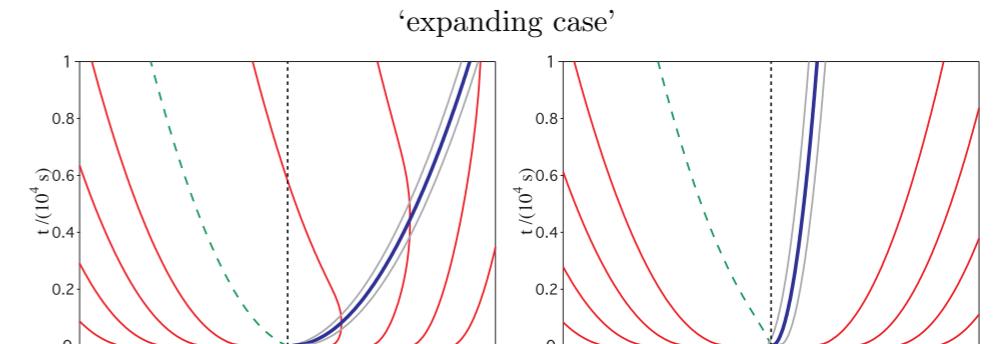
$$M_B = M_0 [\beta_2(1 - X_B) + \beta_1 X_B]$$

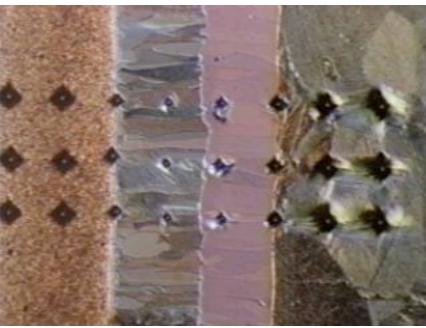
$$v(z, t) = X_A X_B [M_B - M_A] \frac{\partial}{\partial z} \left(-\Omega(X_B - X_A) + RT(\ln X_B - \ln X_A) - 2K \frac{\partial^2 X_B}{\partial z^2} \right)$$

$$\frac{\partial Z(z, t)}{\partial t} = -v \frac{\partial Z(z, t)}{\partial z}$$

both Eulerian and Lagrangian

compared with
sharp interface similarity solution
and DICTRA calculations





W. J. Boettinger, J. E.
C. E. Campbell & G. B. McFadden,
Proceedings of the Royal Society
463 (2007) 334–352

Kirkendall effect

PROCEEDINGS
OF
THE ROYAL
SOCIETY A

Proc. R. Soc. A (2007) **463**, 3347–3373
doi:10.1098/rspa.2007.1904
Published online 9 October 2007

Computation of the Kirkendall velocity and displacement fields in a one-dimensional binary diffusion couple with a moving interface

By W. J. BOETTINGER¹, J. E. GUYER^{1,*}, C. E. CAMPBELL¹
and G. B. MCFADDEN²

Fipy implementation of Cahn-Hilliard model for
“Computation of the Kirkendall Velocity and Displacement Field in a 1-D
Binary Diffusion Couple with a Moving Interface”

W. J. Boettinger¹, J. E. Guyer¹, C. E. Campbell¹, and G. B. McFadden²

¹Metallurgy Division, Materials Science and Engineering Laboratory,
and

²Mathematical and Computational Sciences Division, Information Technology Laboratory,

National Institute of Standards and Technology, Gaithersburg, MD 20899, USA
September 11, 2007

Module kirkendall

Attention!

This software was developed at the National Institute of Standards and Technology by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code this software is not subject to copyright protection and is in the public domain. `kirkendall.py` is an experimental system. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. We would appreciate acknowledgement if the software is used.

This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified.

Fipy implementation of Cahn-Hilliard model for Kirkendall diffusion.

Note

Information about Fipy can be found at <http://www.ctcms.nist.gov/fipy>
This script was tested with revision 2108 of the Fipy source code, available from
<https://www.matforge.org/fipy/browser/trunk>

We solve for the concentration X_B on a 1D domain of length $L = 10$ with 2000 evenly spaced mesh points.

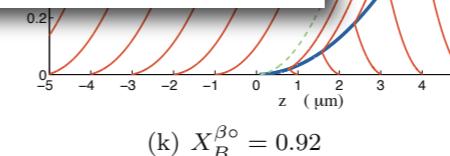
```
>>> from fipy import *
>>> L = 10.
>>> nx = 2000
>>> mesh = Grid1D(nx=nx, dx=L / nx)
>>> xB = CellVariable(name="X_B", mesh=mesh, hasOld=1)
>>> xA = 1. - xB
```

We set the initial conditions of

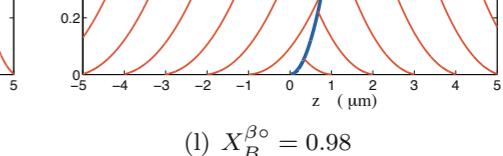
$$X_B = \begin{cases} 0 & \text{for } \zeta \leq L/2 \\ 0.92 & \text{for } \zeta > L/2 \end{cases}$$

where ζ is the position within the domain (at this stage, we don't attach any particular interpretation to the frame of ζ):

1

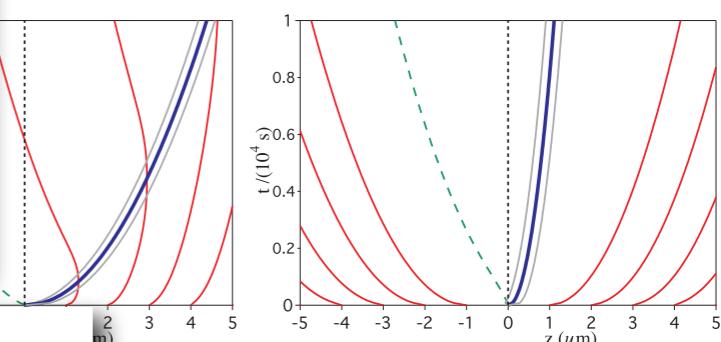


(k) $X_B^{\beta_0} = 0.92$



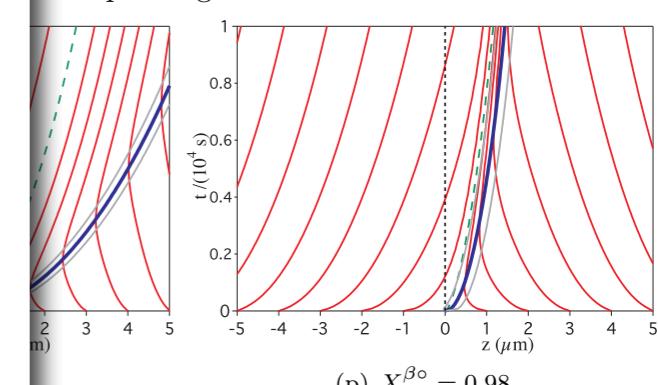
(l) $X_B^{\beta_0} = 0.98$

‘expanding case’



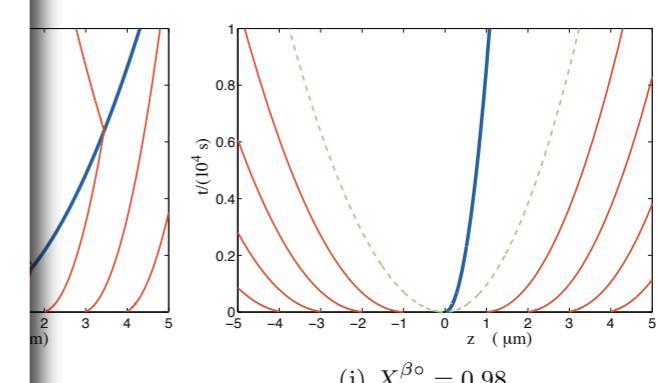
(n) $X_B^{\beta_0} = 0.98$

‘squeezing case’



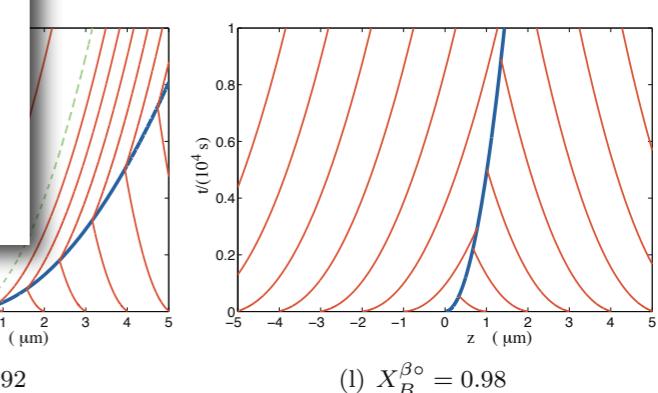
(p) $X_B^{\beta_0} = 0.98$

‘expanding case’



(j) $X_B^{\beta_0} = 0.98$

‘squeezing case’



(i) $X_B^{\beta_0} = 0.98$

$$\frac{\partial X_B}{\partial t} = \frac{\partial}{\partial z} \left\{ \tilde{M}(X_B) \frac{\partial}{\partial z} \left[-\Omega(X_B - X_A) + \frac{1}{2} M_A M_B \right] \right\}$$

$$\tilde{M} = X_A X_B [X_B M_A - X_A M_B]$$

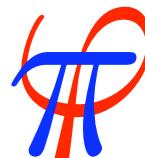
$$M_A = M_0 [\beta_1 (1 - X_A)]^{1/\alpha}$$

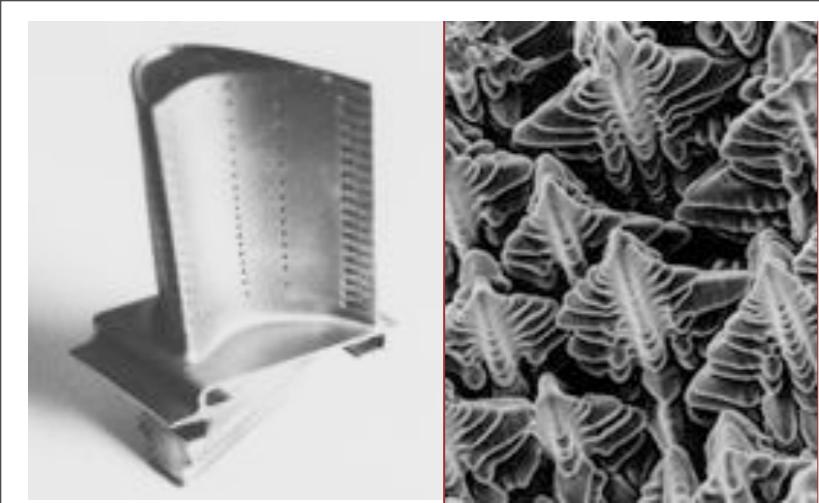
$$M_B = M_0 [\beta_2 (1 - X_B)]^{1/\alpha}$$

$$v(z, t) = X_A X_B [M_B - M_A] \frac{\partial}{\partial z} \left(-\Omega(X_B - X_A) + \frac{1}{2} M_A M_B \right)$$

$$\frac{\partial Z(z, t)}{\partial t} = -v \frac{\partial}{\partial z}$$

both Eulerian and
compared
sharp interface simi-





Phase Field Example

$$\frac{\partial \Delta T}{\partial t} = D_T \nabla^2 \Delta T + \frac{\partial \phi}{\partial t}$$

$$\tau_\phi \frac{\partial \phi}{\partial t} = \nabla \cdot \mathbf{D} \nabla \phi + \phi(1-\phi) \left[\phi - \frac{1}{2} - \frac{\kappa_1}{\pi} \arctan(\kappa_2 \Delta T) \right]$$

$$\beta = \frac{1 - \Phi^2}{1 + \Phi^2}$$

$$\mathbf{D} = \alpha^2 (1 + c\beta) \begin{bmatrix} 1 + c\beta & -c \frac{\partial \beta}{\partial \psi} \\ c \frac{\partial \beta}{\partial \psi} & 1 + c\beta \end{bmatrix} \quad \Phi = \tan\left(\frac{N}{2}\psi\right)$$

$$\psi = \theta + \arctan \frac{\partial \phi / \partial y}{\partial \phi / \partial x}$$



Phase Field Example

```

from fipy import *
dx = dy = 0.025; nx = ny = 500
mesh = Grid2D(dx=dx, dy=dy, nx=nx, ny=ny)
dt = 5e-4

phase = CellVariable(name=r'$\phi$', mesh=mesh, hasOld=True)
dT = CellVariable(name=r'$\Delta T$', mesh=mesh, hasOld=True)
heatEq = (TransientTerm()
    == DiffusionTerm(2.25)
    + (phase - phase.getOld()) / dt)

alpha = 0.015; c = 0.02; N = 6.; theta = pi / 8.
psi = theta + arctan2(phase.getFaceGrad()[1],
    phase.getFaceGrad()[0])
Phi = tan(N * psi / 2)
PhiSq = Phi ** 2
beta = (1. - PhiSq) / (1. + PhiSq)
DbetaDpsi = -N * 2 * Phi / (1 + PhiSq)
Ddia = (1. + c * beta)
Doff = c * DbetaDpsi
D = alpha**2 * (1. + c * beta) * (Ddia * ((1, 0),
    (0, 1)) + Doff * ((0, -1),
    (1, 0)))

tau = 3e-4; kappa1 = 0.9; kappa2 = 20.
phaseEq = (TransientTerm(tau)
    == DiffusionTerm(D)
    + ImplicitSourceTerm((phase - 0.5 - kappa1 / pi * arctan(kappa2 * (1 - phase)))))

radius = dx * 5.
C = (nx * dx / 2, ny * dy / 2)
x, y = mesh.getCellCenters()
phase.setValue(1., where=((x - C[0])**2 + (y - C[1])**2) < radius**2)
dT.setValue(-0.5)

viewer = DendriteViewer(phase=phase, dT=dT,
    title=r"%s & %s" % (phase.name, dT.name),
    datamin=-0.1, datamax=0.05)

for i in range(10000):
    phase.updateOld()
    dT.updateOld()
    phaseEq.solve(phase, dt=dt)
    heatEq.solve(dT, dt=dt)
    if i % 10 == 0:
        viewer.plot()

```

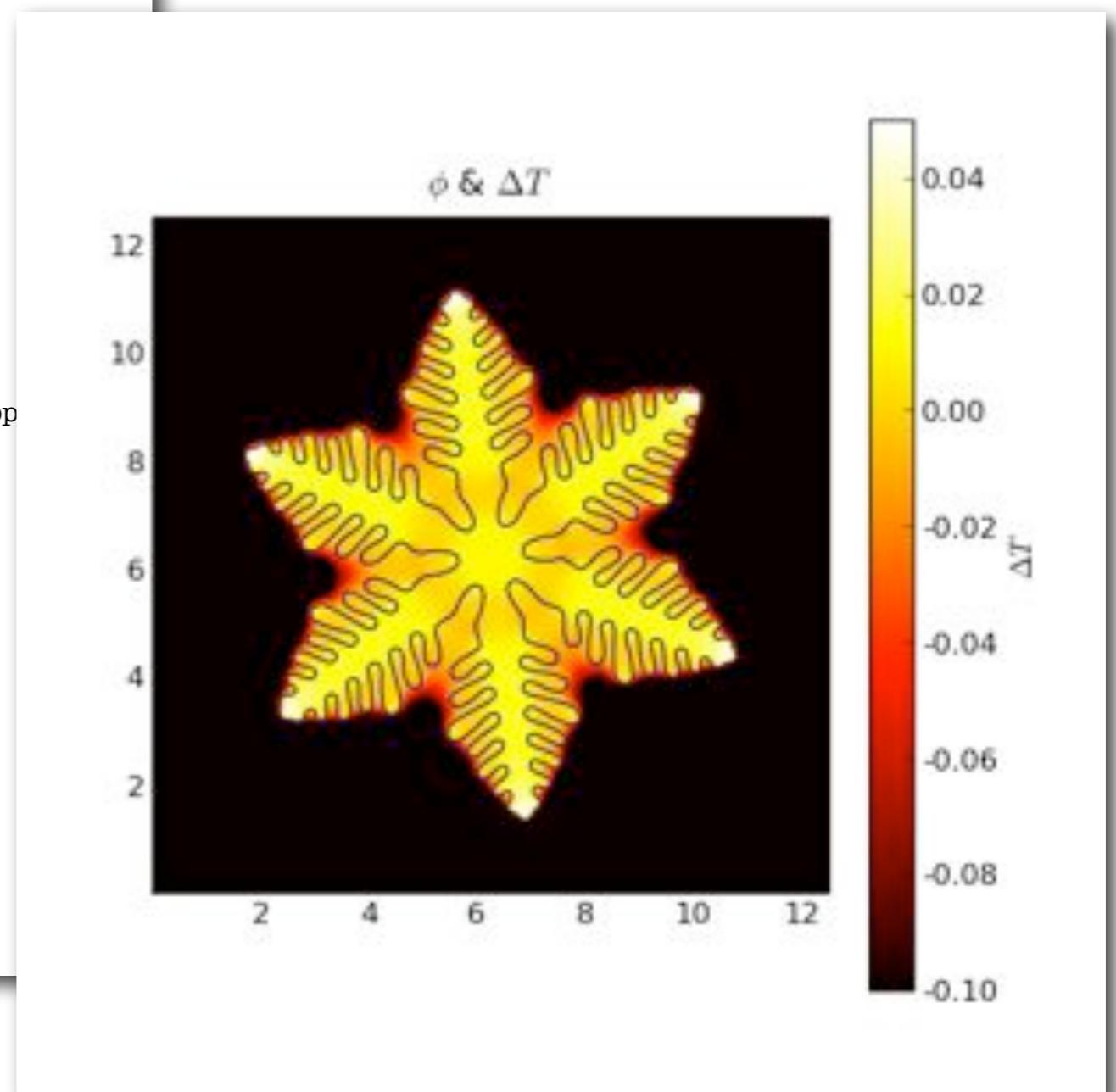
$$\frac{\partial \Delta T}{\partial t} = D_T \nabla^2 \Delta T + \frac{\partial \phi}{\partial t}$$

$$\tau_\phi \frac{\partial \phi}{\partial t} = \nabla \cdot D \nabla \phi + \phi(1 - \phi) \left[\phi - \frac{1}{2} - \frac{\kappa_1}{\pi} \arctan(\kappa_2 \Delta T) \right]$$

$$\beta = \frac{1 - \Phi^2}{1 + \Phi^2}$$

$$\Phi = \tan\left(\frac{N}{2}\psi\right)$$

$$\psi = \theta + \arctan \frac{\partial \phi / \partial y}{\partial \phi / \partial x}$$





Phase Field Example

```

from fipy import *
dx = dy = 0.025; nx = ny = 500
mesh = Grid2D(dx=dx, dy=dy, nx=nx, ny=ny)
dt = 5e-4

phase = CellVariable(name=r'$\phi$', mesh=mesh, hasOld=True)
dT = CellVariable(name=r'$\Delta T$', mesh=mesh, hasOld=True)
heatEq = (TransientTerm()
    == DiffusionTerm(2.25)
    + (phase - phase.getOld()) / dt)

alpha = 0.015; c = 0.02; N = 6.; theta = pi / 8.
psi = theta + arctan2(phase.getFaceGrad()[1],
                       phase.getFaceGrad()[0])

Phi = tan(N * psi / 2)
PhiSq = Phi ** 2
beta = (1 import pylab
DbetaDr class DendriteViewer(Matplotlib2DGridViewer):
Ddia = ( def __init__(self, phase, dT, title=None, limits={}, **kwlimits):
Doff = c self.phase = phase
self.contour = None
D = alpha
tau = 3e
phaseEq

def _plot(self):
    Matplotlib2DGridViewer._plot(self)

radius =
C = (nx
x, y = me
phase.se
dT.setVa
viewer =
for i in r
phase.
self.con
dT.upda
phaseEq.s
heatEq.s
if i % 10 == 0:
    viewer.plot()

```

$$\frac{\partial \Delta T}{\partial t} = D_T \nabla^2 \Delta T + \frac{\partial \phi}{\partial t}$$

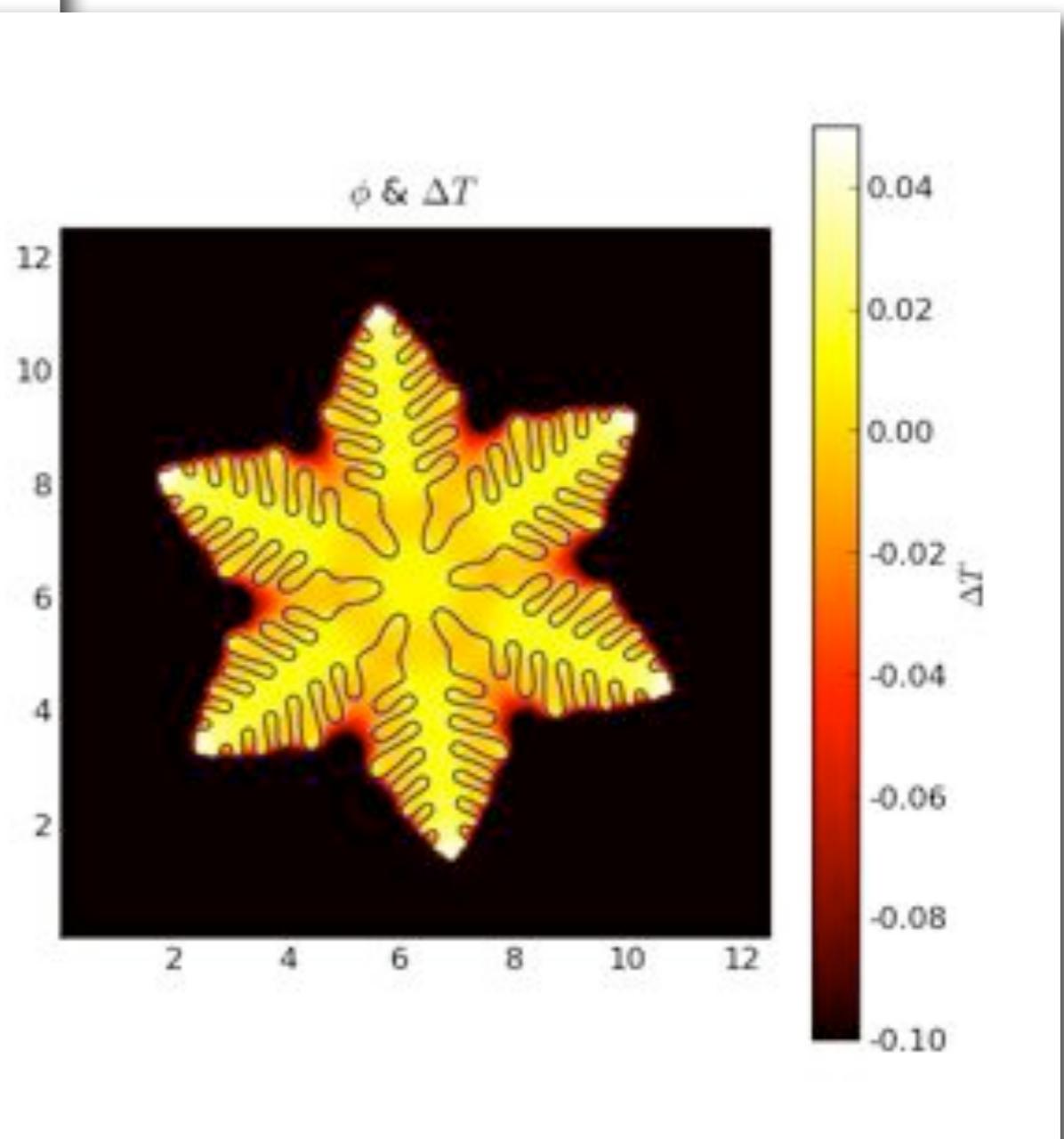
$$\tau_\phi \frac{\partial \phi}{\partial t} = \nabla \cdot D \nabla \phi + \phi(1-\phi) \left[\phi - \frac{1}{2} - \frac{\kappa_1}{\pi} \arctan(\kappa_2 \Delta T) \right]$$

$$\beta = \frac{1 - \Phi^2}{1 + \Phi^2}$$

$$D = \alpha^2 (1 + c\beta) \begin{bmatrix} 1 + c\beta & -c \frac{\partial \beta}{\partial \psi} \\ c \frac{\partial \beta}{\partial \psi} & 1 + c\beta \end{bmatrix}$$

$$\Phi = \tan\left(\frac{N}{2}\psi\right)$$

$$\psi = \theta + \arctan \frac{\partial \phi / \partial y}{\partial \phi / \partial x}$$



“Superfill”

no “leveler”

D. Josell, D. Wheeler, W. H. Huber & T. P. Moffat

Physical Review Letters **87**(1) (2001) 016102

$$v = \frac{\Omega}{nF} (b_0 + b_1 \theta) \frac{c_m^i}{c_m^\infty} \exp\left(\frac{-\alpha F}{RT} \eta\right)$$

$$\frac{\partial \phi}{\partial t} + v_{\text{ext}} |\nabla \phi| = 0$$

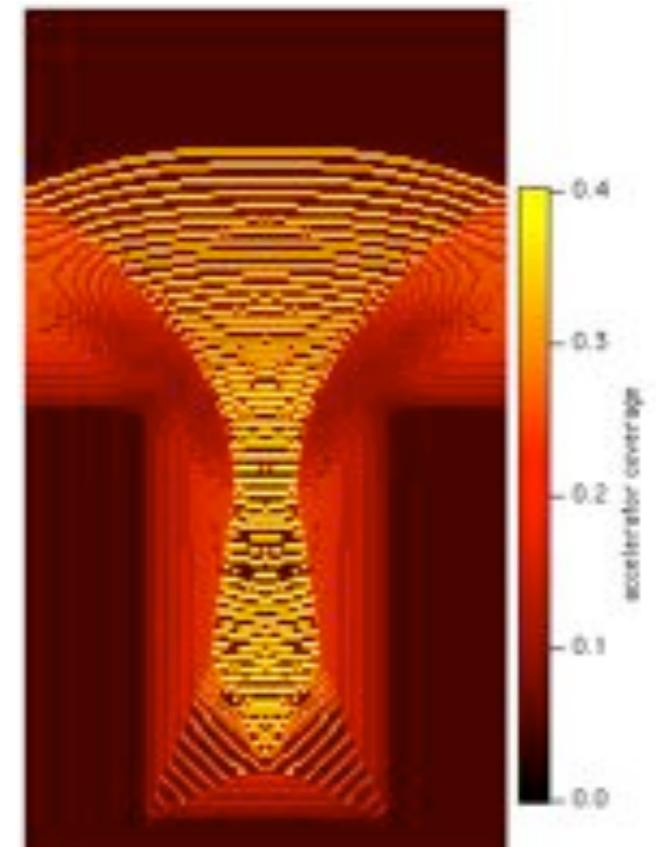
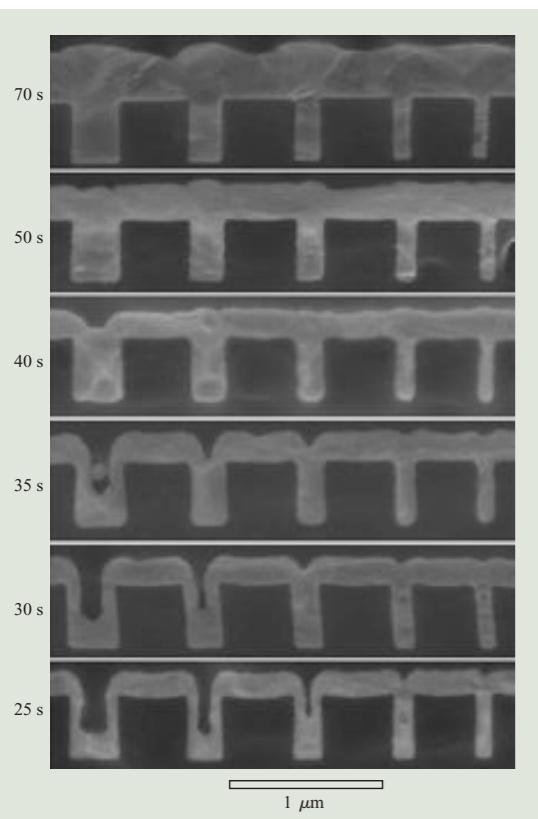
$$\frac{\partial \theta}{\partial t} - Jv\theta - k_\theta c_\theta^i(1-\theta) = 0$$

$$\frac{\partial c_m}{\partial t} - \nabla \cdot D_m \nabla c_m = 0 \quad D_m \hat{n} \cdot \nabla c_m = \frac{v}{\Omega} \quad \text{on } \phi = 0$$

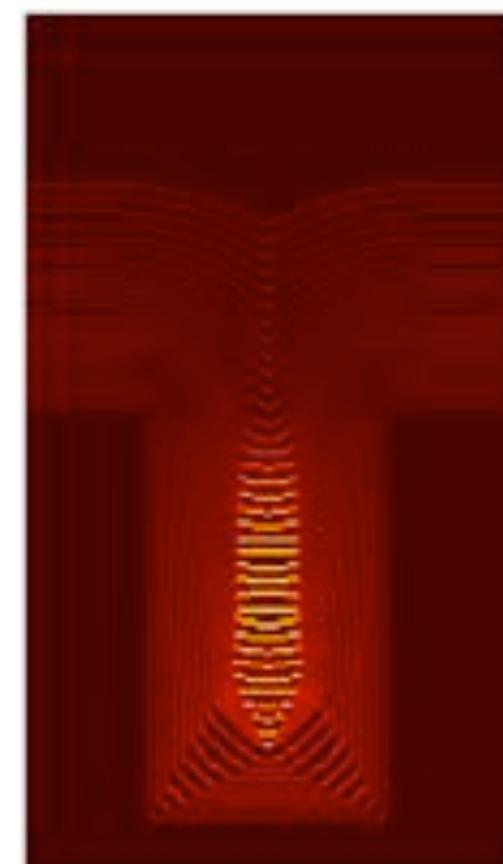
$$\frac{\partial c_\theta}{\partial t} - \nabla \cdot D_\theta \nabla c_\theta = 0 \quad D_\theta \hat{n} \cdot \nabla c_\theta = -k_\theta c_\theta \Gamma(1-\theta) \quad \text{on } \phi = 0$$

$$|\nabla \phi| = 1$$

Level Set method
(combination of PDE
and algorithmic solution)



with “leveler”



Photovoltaics

J. E. Guyer & D. Josell

$$0 = \nabla \cdot \epsilon \nabla V + p - n + N_D^+ - N_A^-$$

$$0 = -\nabla \cdot (\mu_n n \nabla \phi_n) + G - U$$

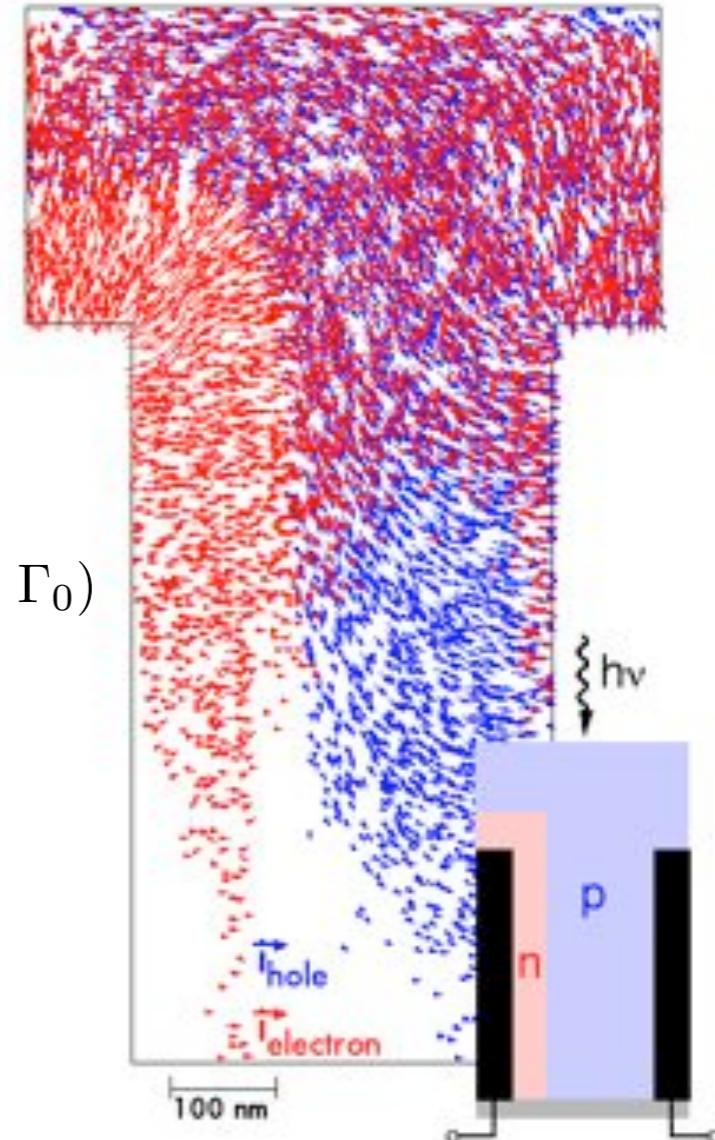
$$0 = -\nabla \cdot (\mu_p p \nabla \phi_p) + G - U$$

$$G = \alpha \Gamma \quad (0 = \nabla \cdot \Gamma \vec{c} + \alpha c \Gamma, \quad \Gamma|_{z=0} = \Gamma_0)$$

$$U = \frac{pn - n_i^2}{\tau_{p0}(n + n') + \tau_{n0}(p + p')}$$

$$n = n_i \exp \left(\frac{V - \phi_n}{kT} \right)$$

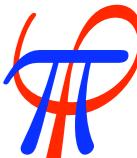
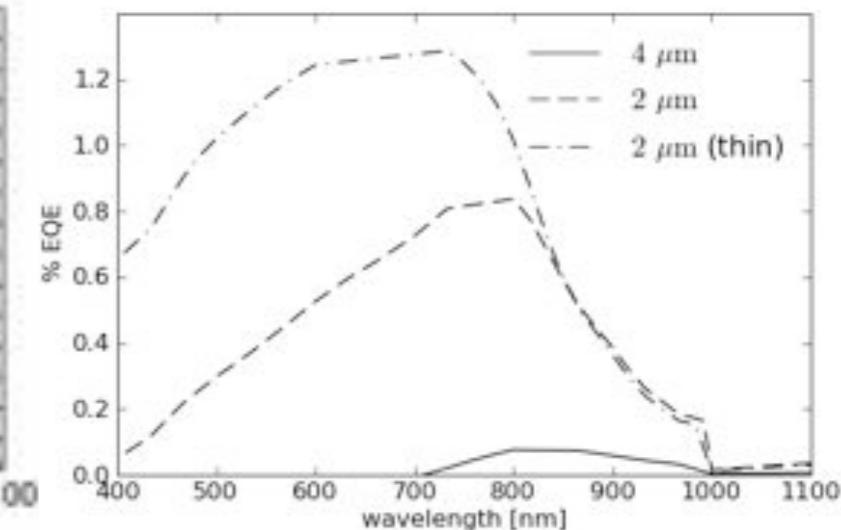
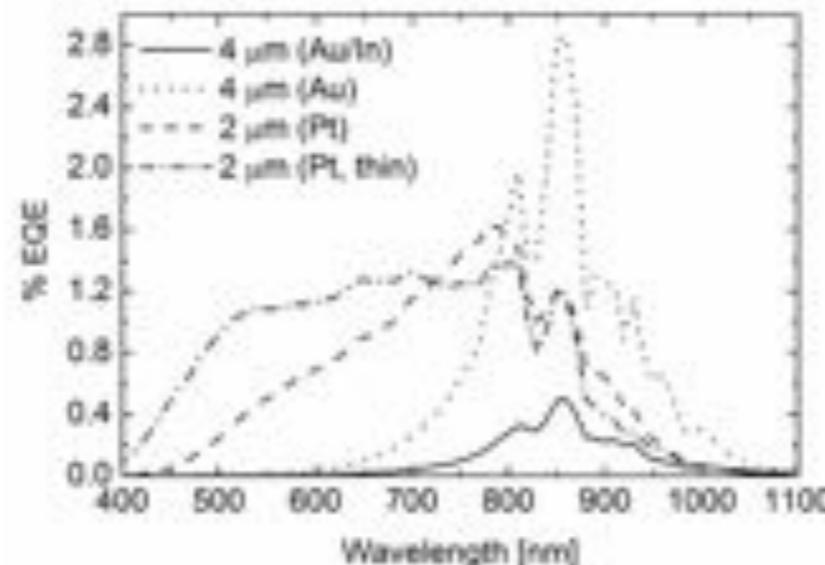
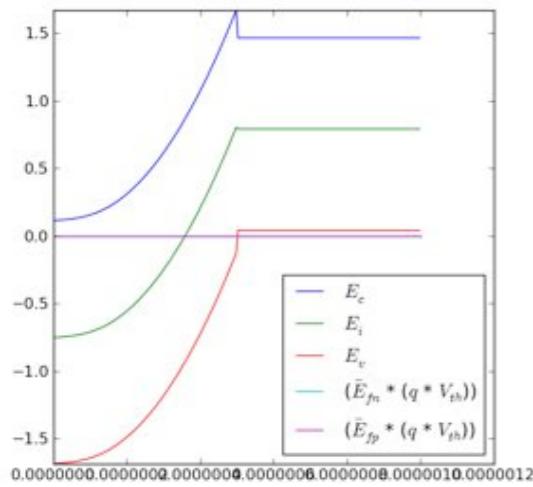
$$p = n_i \exp \left(\frac{\phi_p - V}{kT} \right)$$

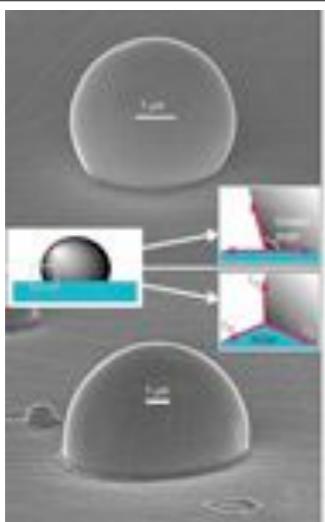


Scharfetter-Gummel discretization

Newton-Block SOR sweeping
measure

model





Reactive Wetting

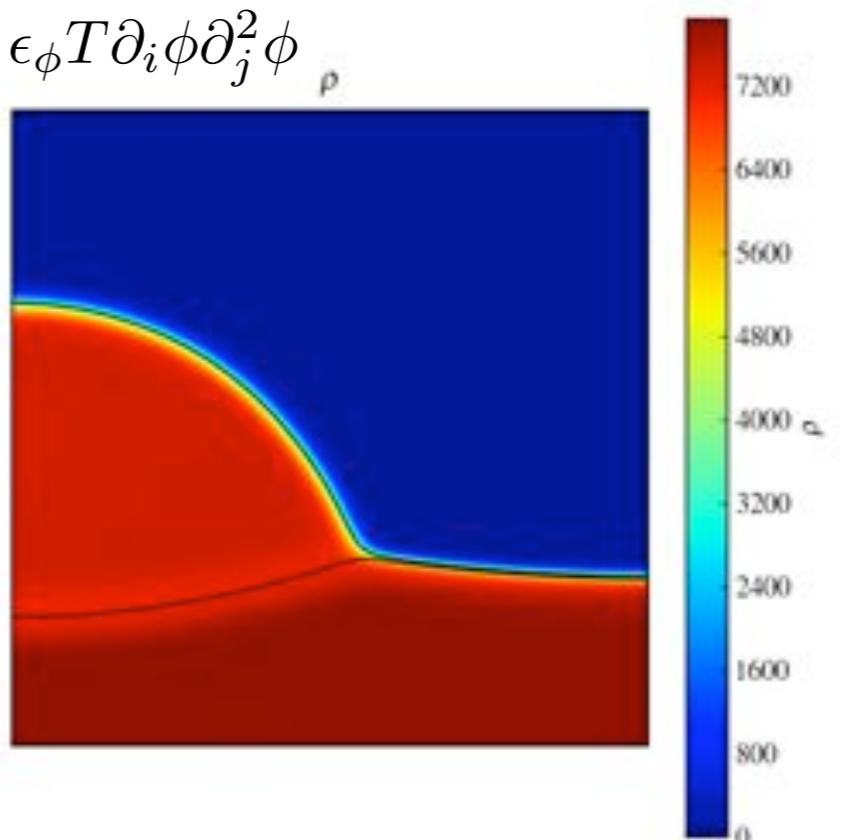
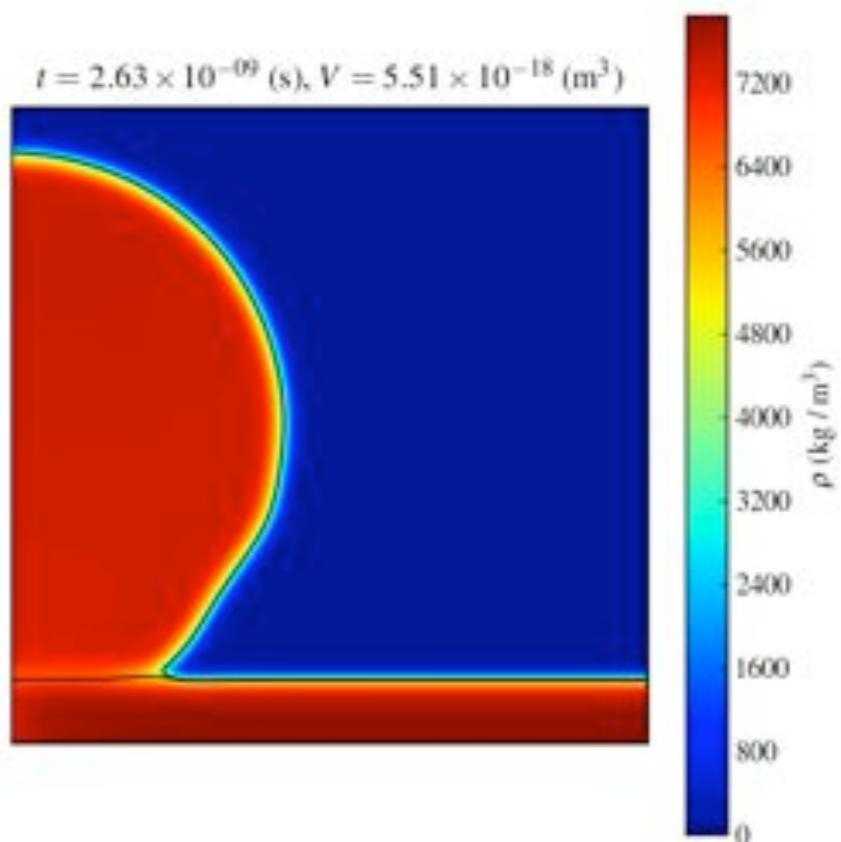
D.Wheeler, J.A.Warren & W.J.Boettigner

$$\frac{\partial \rho_1}{\partial t} + \partial_j (\rho_1 u_j) = \partial_j \left(\frac{M}{T} \partial_j [\mu_1^{NC} - \mu_2^{NC}] \right)$$

$$\frac{\partial \rho_2}{\partial t} + \partial_j (\rho_2 u_j) = \partial_j \left(\frac{M}{T} \partial_j [\mu_2^{NC} - \mu_1^{NC}] \right)$$

$$\begin{aligned} \frac{\partial (\rho u_i)}{\partial t} + \partial_j (\rho u_i u_j) &= \partial_j (\eta [\partial_j u_i + \partial_i u_j]) - \partial_i P \\ &\quad + \epsilon_1 T \rho_1 \partial_i \partial_j^2 \rho_1 + \epsilon_2 T \rho_2 \partial_i \partial_j^2 \rho_2 - \epsilon_\phi T \partial_i \phi \partial_j^2 \phi \end{aligned}$$

$$\frac{\partial \phi}{\partial t} + u_j \partial_j \phi = \epsilon_\phi M_\phi \partial_j^2 \phi - \frac{M_\phi}{T} \frac{\partial F}{\partial \phi}$$



Coupled solution
Parallel

don't invent when you can steal



SciPy



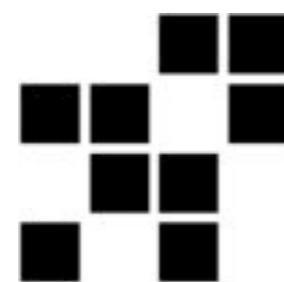
matplotlib



python

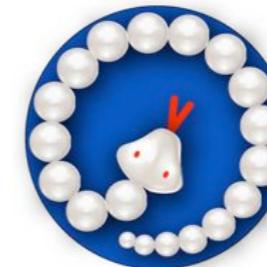


Gmsh



Pysparse

The Trilinos Project



Variable

```
>>> A = Variable(value=3)
>>> B = Variable(value=4)
>>> C = A * B ** 2
>>> C
(Variable(value=array(3)) * (pow(Variable(value=array(4)), 2)))
>>> print C
48
>>> B.setValue(5)
>>> print C
75
```

- Contains, but doesn't inherit from, NumPy array
- MeshVariable holds field of values; Mesh holds geometry and topology
- units (mostly stolen from Konrad Hinsen)

```
>>> (Variable("1 ft*lb*gn") / "2 wk").inUnitsOf('N*m/s')
PhysicalField(1.12088124035e-06,'m*N/s')
```

- supports automatic weave inlining

```
>>> C._getCstring()
'(var0 * (pow(var10, var11)))'
```

Design: test-based development

- hundreds of major tests, comprising thousands of low-level tests
- Tests are documentation (and vice versa)

298

Module `fipy.variables.variable`

`--ge__(self, other)`

Test if a Variable is greater than or equal to another quantity

```
>>> a = Variable(value = 3)
>>> b = (a >= 4)
>>> b
(Variable(value = 3) >= 4)
>>> b()
0
>>> a.setValue(4)
>>> b()
1
>>> a.setValue(5)
>>> b()
1
```

Design: test-based development

- hundreds of major tests, comprising thousands of low-level tests
- Tests are documentation (and vice versa)

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)))

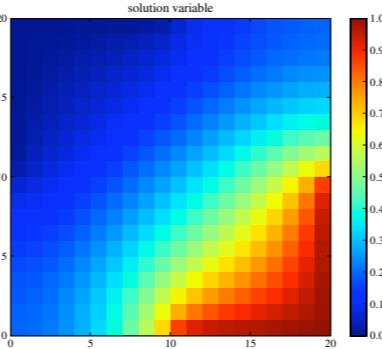
>>> BCs = (FixedValue(faces=facesTopLeft, value=valueTopLeft),
...          FixedValue(faces=facesBottomRight, value=valueBottomRight))
```

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(var=phi,
...              boundaryConditions=BCs,
...              dt=timeStepDuration)
...     if __name__ == '__main__':
...         viewer.plot()
```



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

Overview

http://www.ctcms.nist.gov/fipy/

NIST Time | NIST Home | About NIST | Contact Us | A-Z Site Index | Search

About MSEL | Topic/Subject Areas | Products/Services | News/Multimedia | Programs/Projects | Facilities

NIST Home > MSEL > CTCMS > FiPy: A Finite Volume PDE Solver Using Python

FiPy: A Finite Volume PDE Solver Using Python

Home

- Download
- Installation
 - for Mac OS X
 - for Windows
 - via SVN
- Manual (PDF)
- Reference (PDF)
- Mailing List

FiPy@MatForge

- Source code
- Bugs, etc.

Applications

- Superconformal electrodeposition
- Virtual Kinetics of Materials Laboratory

Working Group

- Jonathan Guyer
- Daniel Wheeler
- James Warren

Administrata

- Credits
- License
- Disclaimer
- Privacy / Security / Accessibility

Overview

FiPy is an object oriented, partial differential equation (PDE) solver, written in Python, based on a standard finite volume (FV) approach. The framework has been developed in the Metallurgy Division and Center for Theoretical and Computational Materials Science (CTCMS), in the Materials Science and Engineering Laboratory (MSEL) at the National Institute of Standards and Technology (NIST).

The solution of coupled sets of PDEs is ubiquitous to the numerical simulation of science problems. Numerous PDE solvers exist, using a variety of languages and numerical approaches. Many are proprietary, expensive and difficult to customize. As a result, scientists spend considerable resources repeatedly developing limited tools for specific problems. Our approach, combining the FV method and Python, provides a tool that is extensible, powerful and freely available. A significant advantage to Python is the existing suite of tools for array calculations, sparse matrices and data rendering.

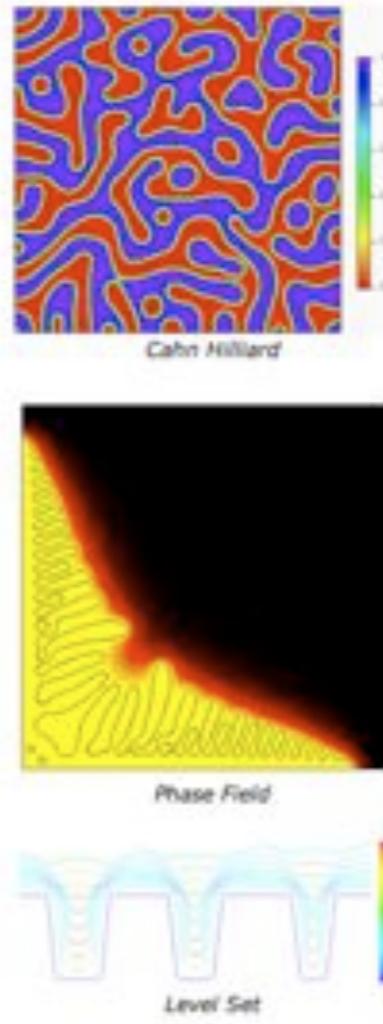
The FiPy framework includes terms for transient diffusion, convection and standard sources, enabling the solution of arbitrary combinations of coupled elliptic, hyperbolic and parabolic PDEs. Currently implemented models include phase field treatments of polycrystalline, dendritic, and electrochemical phase transformations as well as a level set treatment of the electrodeposition process.

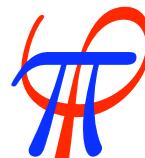
Even if you don't read manuals...
...please read the Installation Guide and Chapter 6: "Frequently Asked Questions" of the Manual.

What's new in version 2.0.2?

Warning

FiPy 2 brings unavoidable syntax changes. Please see [example.updated_usage_2](#) for guidance on the changes that you will need to make to your FiPy 1.x scripts.





fipy@nist.gov

Exploding residuals in convection-diffusion equation

9 Feb 23:36 Jonathan Guyer □ Version 2.0
6 Feb 15:41 Pacif □ Installation question
9 Feb 19:56 Daniel Wheeler □ Installation question
10 Feb 02:44 Pacif □ Installation question
10 Feb 15:56 Jonathan Guyer
10 Feb 16:36 Daniel Wheeler
7 Feb 15:56 Etienne Rivard □ Exploding residuals in convection-diffusion equation
7 Feb 17:44 Jonathan Guyer □ Exploding residuals in convection-diffusion equation
8 Feb 17:27 Etienne Rivard □ Exploding residuals in convection-diffusion equation
10 Feb 00:04 Jonathan Guyer □ Exploding residuals in convection-diffusion equation
9 Feb 16:30 Etienne Rivard □ Exploding residuals in convection-diffusion equation
9 Feb 17:38 Etienne Rivard □ Exploding residuals in convection-diffusion equation
7 Feb 05:50 fred2 □ stable/current/trunk version + doc?

From: Etienne Rivard <etienne.rivard@...>
Subject: Exploding residuals in convection-diffusion equation
Newsgroups: gmane.comp.python.fipy
Date: 2009-02-07 14:58:45 GMT (3 days, 7 hours and 26 minutes ago)

Hello everyone,

I am trying to solve the following equation:

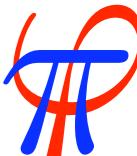
```
\n\\frac{\\partial }{\\left( \\partial t \\right)^2} - \\frac{\\partial }{\\left( \\partial x \\right)}\n+ 6x\\frac{\\partial }{\\left( \\partial x \\right)^2} = 0\n\\frac{\\partial }{\\left( \\partial x \\right)} = \\frac{\\partial }{\\left( \\partial t \\right)^2} + 6x\\frac{\\partial }{\\left( \\partial x \\right)^2}\n\\frac{\\partial }{\\left( \\partial x \\right)} = \\frac{\\partial }{\\left( \\partial t \\right)^2} + 6x\\frac{\\partial }{\\left( \\partial x \\right)^2}
```

Sorry for the messy code, but this is latex output from MathType. I would suggest pasting it in your favorite latex compiler.

I coded the equation this way:

```
diffTerm = ImplicitDiffusionTerm(coeff=sigma**2.0*nu_p/8**2.0)

eq1 = TransientTerm(coeff=sigma**2.0*nu) \
```



Build Configuration "trunk" - Fipy - Trac

<http://www.matforge.org/Fipy/build/trunk?page=5>

 **Fipy: A Finite Volume PDE Solver Using Python**

[Fipy Home](#) [Login](#) [Help/Guide](#) [About Trac](#) [MatForge Home](#) [Participate](#) [Preferences](#)

[Wiki](#) [Timeline](#) [Roadmap](#) [Browse Source](#) [View Tickets](#) [New Ticket](#) [Search](#) [Downloader](#) [Build Status](#)

[-- Previous Page](#) [Next Page --](#)

Build Configuration "trunk"

Repository path: [trunk](#)

Fipy trunk.. Should be stable.

Chugnet	Linux	Mac OS X	Mac OS X - Trunk	Windows	sandbox
[29911]	-	-	-	-	-
[29910]	5 / 297: Failed	5 / 298: Failed	5 / 299: Failed	800: Success	5 / 291: Failed
	diff	patch	patch	patch	patch
	Log				
	Changeset				
	Test				
	Setup				
	Precompile				
	Test				
	Trilinos				
	Matrix				
	Linear				
	Diffusion				
	Convection				
	Reaction				
	Transport				
	Chemical				
	Electro				
	Thermal				
	Structural				
	Optimization				
	Solvers				
	Utilities				
	Examples				
	Documentation				
	API				
	Tests				
	Coverage				
	Profile				
	Timing				
	Memory				
	Performance				
	Parallel				
	MPI				
	OpenMP				
	OpenCL				
	CUDA				
	HPC				
	GPU				
	MPI4Py				
	Trilinos4Py				
	PyOP2				
	PyAMG				
	PyLAPACK				
	PyBLAS				
	PySLEIGN2				
	PyTBB				
	PyPETSc				
	PyDOLFIN				
	PyFEniCS				
	PyDOLFIN++				
	PyFEniCS++				
	PyDOLFINX				
	PyFEniCSX				
	PyDOLFIN++X				
	PyFEniCS++X				
	PyDOLFINX++				
	PyFEniCSX++				
	PyDOLFINX++X				
	PyDOLFINX++X++				
	PyFEniCSX++X++				
	PyDOLFINX++X++X				
	PyFEniCSX++X++X				
	PyDOLFINX++X++X++				
	PyFEniCSX++X++X++				
	PyDOLFINX++X++X++X				
	PyFEniCSX++X++X++X				
	PyDOLFINX++X++X++X++				
	PyFEniCSX++X++X++X++				
	PyDOLFINX++X++X++X++X				
	PyFEniCSX++X++X++X++X				
	PyDOLFINX++X++X++X++X++				
	PyFEniCSX++X++X++X++X++				
	PyDOLFINX++X++X++X++X++X				
	PyFEniCSX++X++X++X++X++X				
	PyDOLFINX++X++X++X++X++X++				
	PyFEniCSX++X++X++X++X++X++				
	PyDOLFINX++X++X++X++X++X++X				
	PyFEniCSX++X++X++X++X++X++X				
	PyDOLFINX++X++X++X++X++X++X++				
	PyFEniCSX++X++X++X++X++X++X++				
	PyDOLFINX++X++X++X++X++X++X++X				
	PyFEniCSX++X++X++X++X++X++X++X				
	PyDOLFINX++X++X++X++X++X++X++X++				
	PyFEniCSX++X++X++X++X++X++X++X++				
	PyDOLFINX++X++X++X++X++X++X++X++X				
	PyFEniCSX++X++X++X++X++X++X++X++X	PyFEniCSX++X++X++X++X++X++X++X++X	PyFEniCSX++X++X++X++X++X++X++X++X	PyFEniCSX++X++X++X	

A Sampling of External Users

- NIST – gold nanoparticle hyperthermia
- U. of Central Florida – thermomigration and precipitate growth
- U. of Maine – water percolation through peat bogs
- Silesian University of Technology – solar irradiation of soil
- U. C. Berkeley – mRNA reaction-diffusion in fruitfly embryos
- Seagate Technologies – heat conduction in magnetic devices
- Princeton Satellite Systems – wind turbine development
- Rensselaer Polytechnic Institute – fuel cells
- Purdue University – stresses in photodiodes

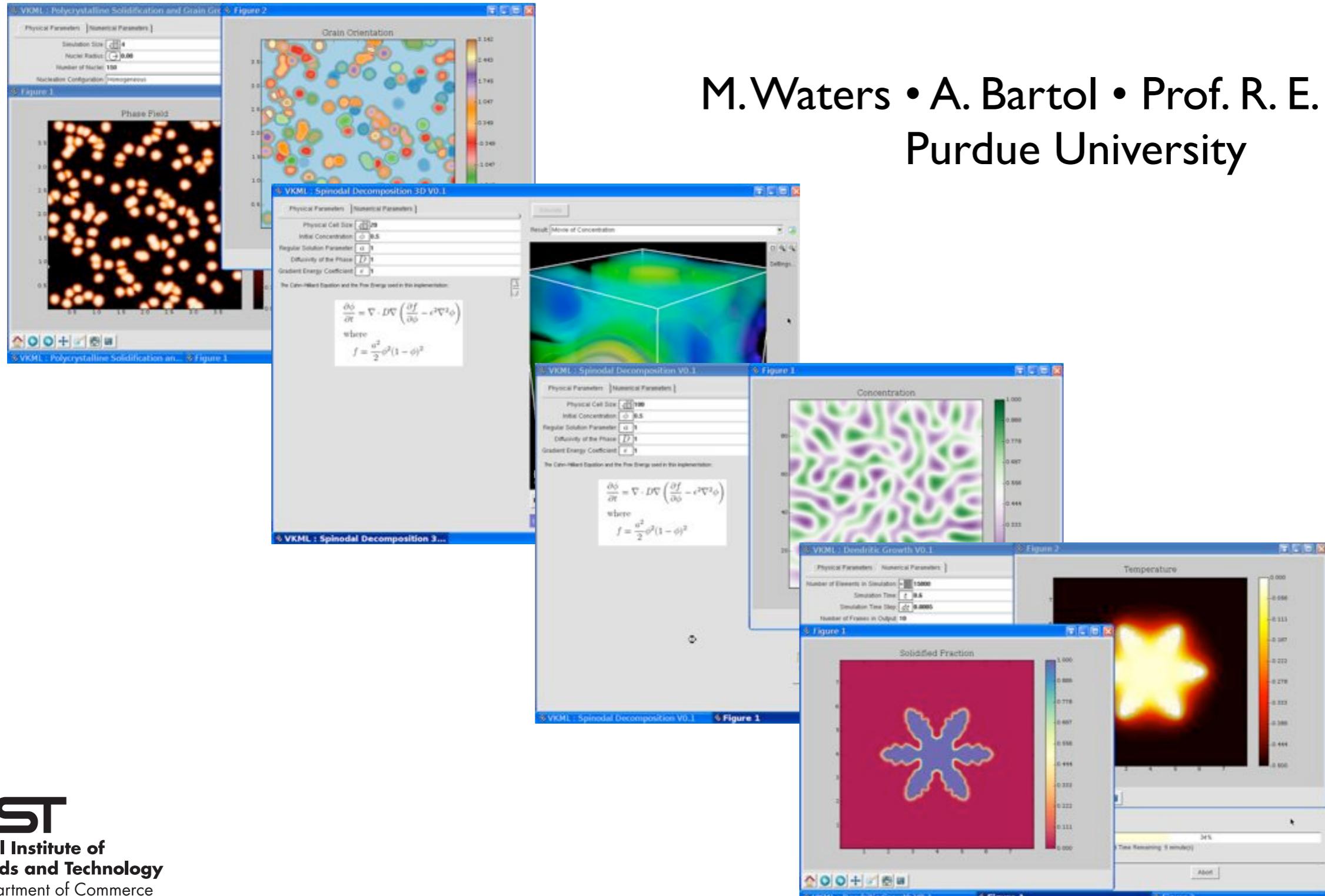
Virtual Kinetics of Materials Library

nanohub.org/tools/vkmlggs

nanohub.org/tools/vkmlpsgg

nanohub.org/tools/vkmlsd

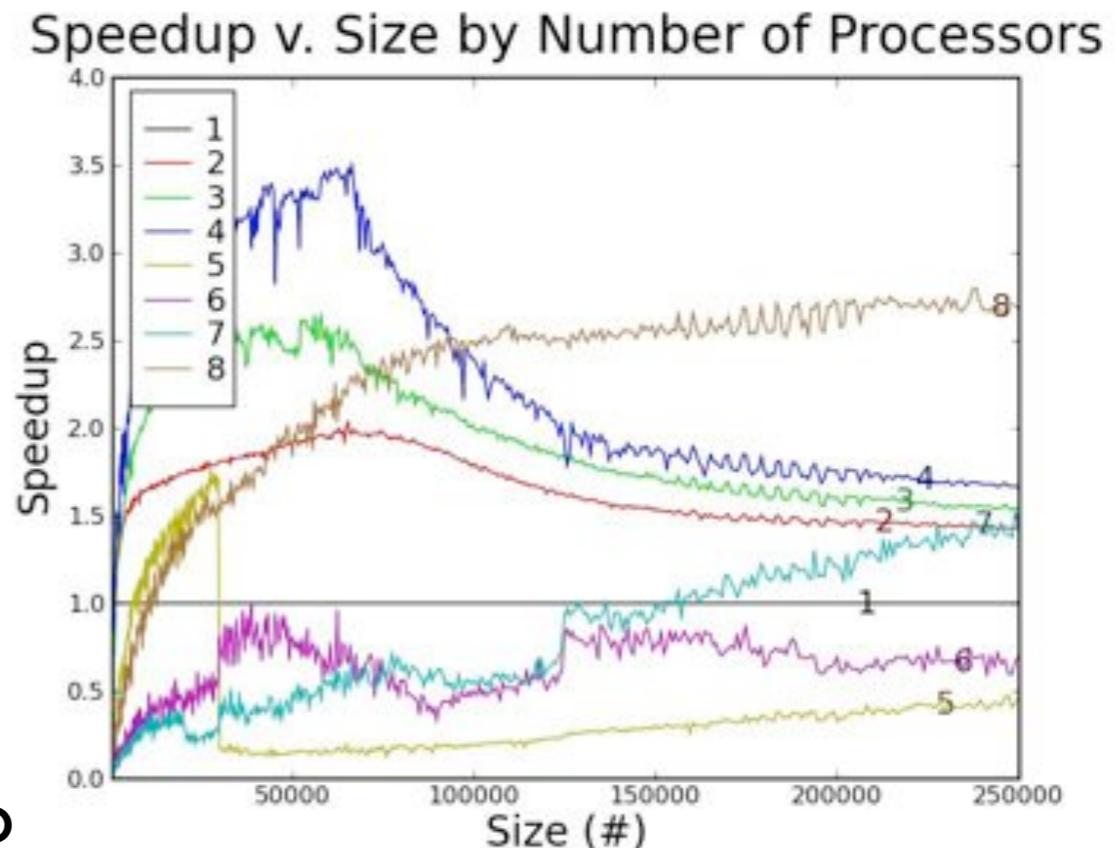
nanohub.org/tools/vkmlsd3d



M. Waters • A. Bartol • Prof. R. E. García
Purdue University

Parallel

- 2007 Max Gibiansky - Harvey Mudd - SURF student
 - Integrate FiPy with PyTrilinos
Matrix solves in parallel...
...but builds in serial
- 2008 Olivia Buzek - U of Maryland - SURF student
Daniel Stiles - Montgomery Blair High School
 - Build matrix in parallel with PyTrilinos, too
 - Pervasive changes to every FiPy data type to make “parallel aware”
 - Fragile and poor scaling performance
- Presently
 - Leave FiPy alone (mostly)
 - Break processes into separate FiPy problems with limited overlap
 - PyTrilinos (and a little mpi4py) for communication and parallel solution

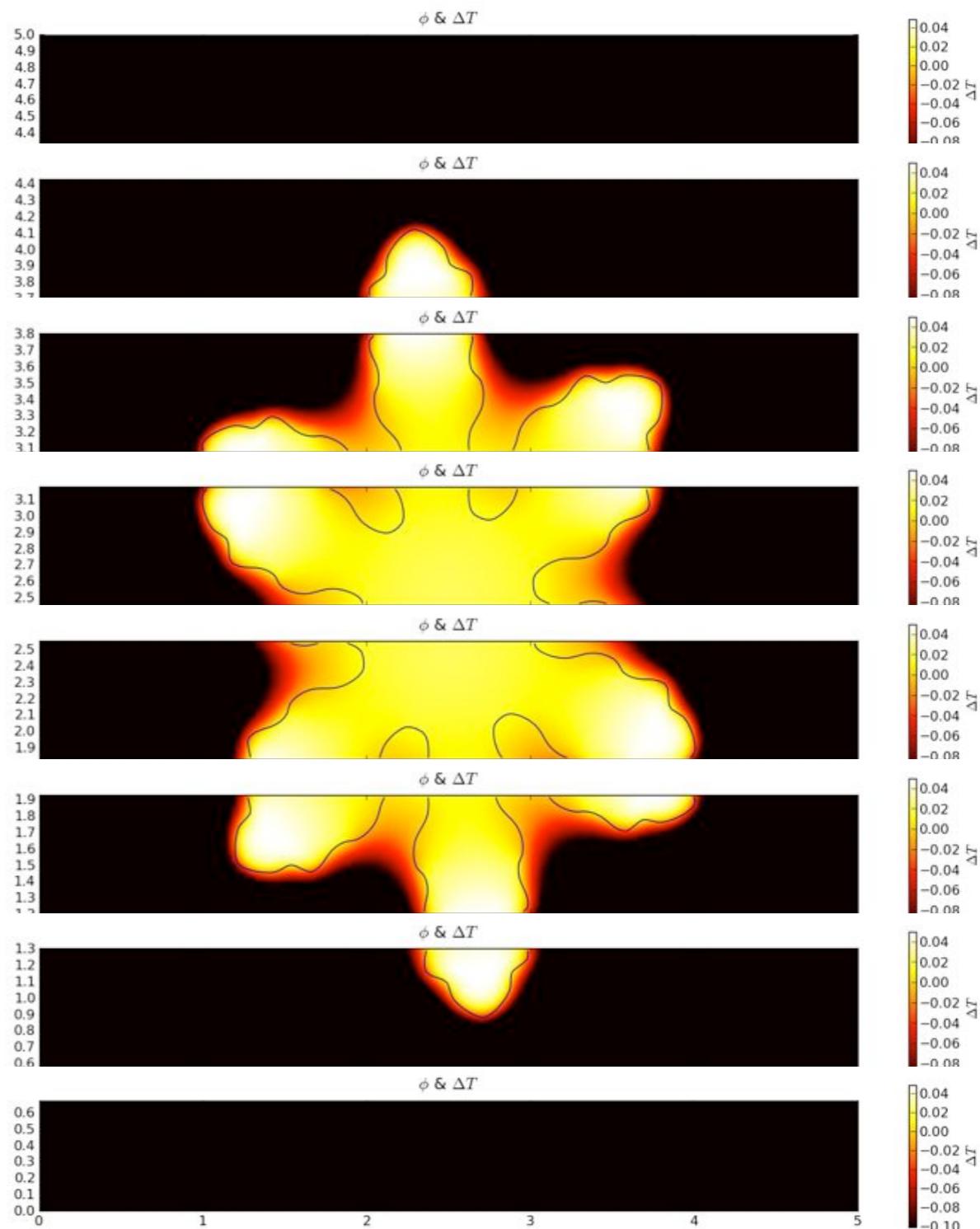
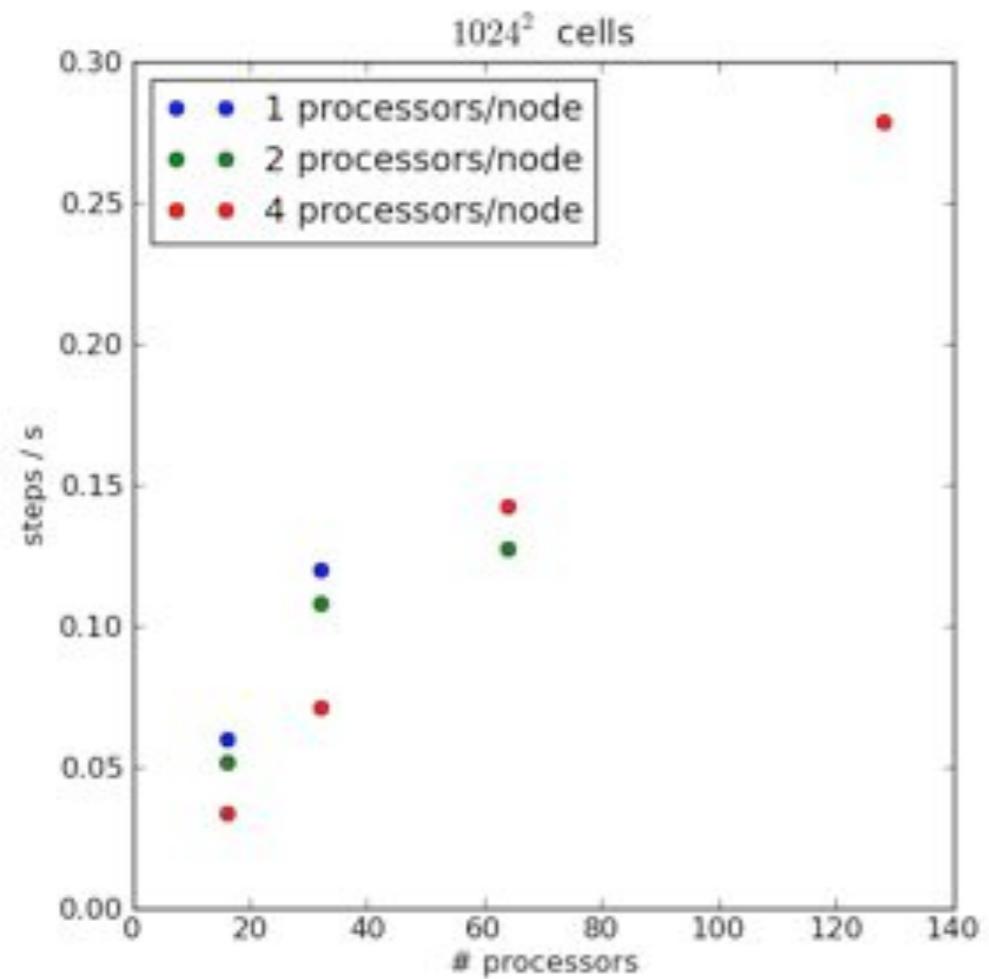


Parallel

mpirun -np 8 python examples/phase/anisotropy.py

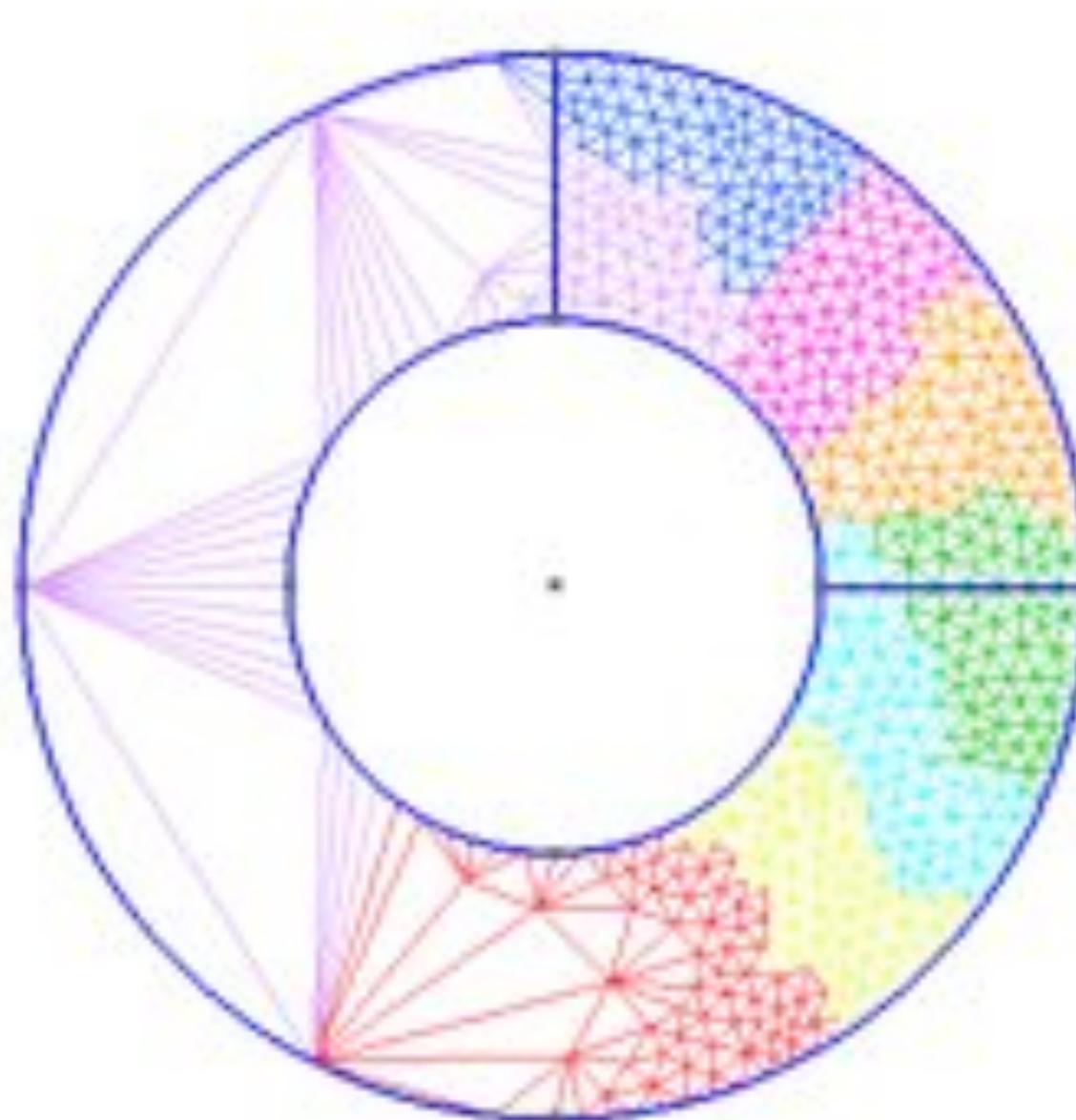
- presently a Trilinos & PySparse hybrid
- doctests
 - conditional tests?
- zero-element arrays

[numpy:ticket:1171](#)



Parallel: Partitioning

- Presently only support primitive slices of Grid meshes
- PyMetis - “hard to install”
- Zoltan (Trilinos) - no Python wrapper
- Gmsh (uses Metis) - easy to install and use, but how to efficiently obtain overlaps?
- NetworkX?



Future Work: Coupled Equations

$$\frac{\partial A}{\partial t} = \nabla \cdot D_{AA} \nabla A + \nabla \cdot D_{AB} \nabla B$$
$$\frac{\partial B}{\partial t} = \nabla \cdot D_{BA} \nabla A + \nabla \cdot D_{BB} \nabla B$$

- Now

```
eqA = TransientTerm() == DiffusionTerm(coeff=DAA) + (DAB * B.getFaceGrad()).getDivergence()  
eqB = TransientTerm() == (DBA * A.getFaceGrad()).getDivergence() DiffusionTerm(coeff=DBB)  
for step in range(steps):  
    resA = resB = 1e10  
    while resA > 1e-3 or resB > 1e-3:  
        resA = eqA.sweep(var=A)  
        resB = eqB.sweep(var=B)
```

- Soon

```
eqA = TransientTerm()(A) == DiffusionTerm(coeff=DAA)(A) + DiffusionTerm(coeff=DAB)(B)  
eqB = TransientTerm()(B) == DiffusionTerm(coeff=DBA)(A) + DiffusionTerm(coeff=DBB)(B)  
eqs = CoupledEquations((eqA, eqB))  
for step in range(steps):  
    eqs.solve(vars=(A, B))
```

Future Work

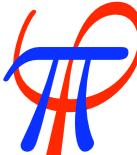
- Solvers
 - SciPy sparse
 - PyAMG - they've already done it!
- FEMhub
- Sphinx
- VisIt
- Autonomous viewers
- CorePy? Cython?

Summary

- Cross-platform, Open Source code for phase transformation problems
- Capable of solving multivariate, coupled, non-linear PDEs
- Extensive documentation, dozens of examples, hundreds of tests
- Object-oriented structure easy to adapt to unique problems
- Slower to run than hand-tailored FORTRAN or C...
 - ...but much faster to write
- Python is great, but...
 - ...scientific Python community is our real enabling technology

FiPy 2.0 - released 2/9/2009
www.ctcms.nist.gov/fipy

“FiPy: PDEs with Python”
J. E. Guyer, D. Wheeler & J.A. Warren
Computing in Science and Engineering **May/June** (2009) 6



Major Contributors

- Alex Mont - Montgomery Blair High School
 - Gmsh import
- Katie Travis - Smith College
 - automated weave inlining
- Max Gibiansky - Harvey Mudd College
 - Trilinos solvers
- Andrew Reeve - U of Maine
 - anisotropic diffusion
- Olivia Buzek - U of Maryland
 - parallel
- Daniel Stiles - Montgomery Blair High School
 - parallel
 - Mayavi2
 - Gmsh partitioning

random thoughts on scientific Python

- Installation much easier, but...
- Be careful about licenses
- Don't just wrap
- Python is a great cross-platform equalizer, but...
 - GPUs are very platform specific
- Embrace 1.0!
- What becomes of SciPy if Python “dies”?
 - What if it's our fault?