Workshop Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morning</td>
<td>Introduction to CIG and PyLith</td>
</tr>
<tr>
<td></td>
<td>Example: Fault in a box</td>
</tr>
<tr>
<td></td>
<td>Break</td>
</tr>
<tr>
<td></td>
<td>Tinker time</td>
</tr>
<tr>
<td></td>
<td>Lunch</td>
</tr>
<tr>
<td></td>
<td>Troubleshooting tips</td>
</tr>
<tr>
<td></td>
<td>Example: 2-D subduction zone</td>
</tr>
<tr>
<td>Afternoon</td>
<td>Break</td>
</tr>
<tr>
<td></td>
<td>Tinker time</td>
</tr>
</tbody>
</table>

Introduction
What is CIG?
Computational Infrastructure for Geodynamics (www.geodynamics.org)

Objective: Develop, support, and disseminate software for the geodynamics community.

- Coordinated effort to develop reusable, well-documented, open-source geodynamics software
- Strategic partnerships with the larger world of computational science and geoinformatics
- Specialized training and workshops for both geodynamics and larger Earth-science communities

Underlying principle: Earth scientists need help from computational scientists to develop state-of-the-art modeling codes
CIG: Institution-Based Organization
Educational and not-for-profit organization

- **Open-organization**
  - Any institution seeking to collaborate on the development of open-source geodynamics software
  - No cost or size requirements

- **Current members**
  - 50 member institutions
  - 10 foreign affiliates

- **NSF funding Jul 2010 – Jun 2015**
CIG Working Groups
Organized by sub-disciplines

- Short-term tectonics
- Long-term tectonics
- Mantle convection
- Computational seismology
- Geodynamo
- Magma dynamics
**Objective:** Simulate crustal deformation across spatial scales from 1 m to $10^3$ km and temporal scales ranging from 0.01 s to $10^5$ years.

- Formed through efforts by Brad Hager and Mark Simons before CIG started
- Strong connection to SCEC Crustal Deformation Modeling focus group
- Building connections with SCEC Earthquake Source Physics focus group
CIG Organizational Structure

- **Staff**
  - Responsible for software development
  - Director handles day-to-day decisions

- **Science Steering Committee**
  - Voice of geophysics community
  - Prioritizes the competing needs of all sub-disciplines

- **Executive Committee**
  - Primary decision-making body
  - Approves SSC recommendations and contractual arrangements

- **Member institution representatives**
  - Vote on membership applications and bylaws

- **Community members**
  - Collaborate with staff to develop software
CIG Activities

- Software development: primary activity
- Workshops
  - Sponsors workshops organized by one or more working groups
  - Holds workshops focusing on scientific computing and geodynamics
- Training in use of CIG software
  - Tutorials at workshops
  - Specialized training sessions (like this one)
- Web site: geodynamics.org
  - Distribution of software and documentation
  - Mailing lists for each working group
  - Wiki-like web pages for community involvement
CIG Software for Crustal Deformation

- **PyLith**
  - Solves 2-D and 3-D problems associated with earthquake faulting and quasi-static and dynamic viscoelastic deformation
  - Short-term tectonics where geometry does not change significantly

- **Gale**
  - Solves problems in orogenesis, rifting, and subduction, including free surfaces with coupling to surface erosion models
  - Long-term tectonics where geometry changes significantly
Quasistatic modeling associated with earthquakes

- Strain accumulation associated with interseismic deformation
  - What is the stressing rate on faults X, Y, and Z?
  - Where is strain accumulating in the crust?
- Coseismic stress changes and fault slip
  - What was the slip distribution in earthquake A?
  - How did earthquake A change the stresses on faults X, Y, and Z?
- Postseismic relaxation of the crust
  - What rheology is consistent with observed postseismic deformation?
  - Can aseismic creep or afterslip explain the deformation?
Dynamic modeling associated with earthquakes

- Modeling of strong ground motions
  - Forecasting the amplitude and spatial variation in ground motion for scenario earthquakes
- Coseismic stress changes and fault slip
  - How did earthquake A change the stresses on faults X, Y, and Z?
- Earthquake rupture behavior
  - What fault constitutive models/parameters are consistent with the observed rupture propagation in earthquake A?
Crustal Deformation Modeling
Elasticity problems where geometry does not change significantly

Volcanic deformation associated with magma chambers and/or dikes

- **Inflation**
  - What is the geometry of the magma chamber?
  - What is the potential for an eruption?

- **Eruption**
  - Where is the deformation occurring?
  - What is the ongoing potential for an eruption?

- **Dike intrusions**
  - What the geometry of the intrusion?
PyLith

- Developers
  - Brad Aagaard (USGS, lead developer)
  - Charles Williams (GNS Science, formerly at RPI)
  - Matthew Knepley (Univ. of Chicago, formerly at ANL)

- Combined dynamic modeling capabilities of EqSim (Aagaard) with the quasistatic modeling capabilities of Tecton (Williams)

- Use modern software engineering (modular design, testing, documentation, distribution) to develop an open-source, community code
Crustal Deformation Modeling
Overview of workflow for typical research problem

Legend

- **CIG**
- **Free**
- **Open Source**
- **Commercial**

- **Available**
- **Planned**

PyLith
Overview
Governing Equations

Elasticity equation

\[ \sigma_{ij,j} + f_i = \rho \ddot{u} \text{ in } V, \quad \text{(1)} \]
\[ \sigma_{ij} n_j = T_i \text{ on } S_T, \quad \text{(2)} \]
\[ u_i = u_i^0 \text{ on } S_u, \text{ and } \]
\[ R_{ki}(u_i^+ - u_i^-) = d_k \text{ on } S_f. \quad \text{(4)} \]

Multiply by weighting function and integrate over the volume,

\[ -\int_V \left(\sigma_{ij,j} + f_i - \rho \ddot{u}_i\right) \phi_i \, dV = 0 \quad \text{(5)} \]

After some algebra,

\[ -\int_V \sigma_{ij} \phi_{i,j} \, dV + \int_{S_T} T_i \phi_i \, dS + \int_V f_i \phi_i \, dV - \int_V \rho \ddot{u}_i \phi_i \, dV = 0 \quad \text{(6)} \]
Writing the trial and weighting functions in terms of basis (shape) functions,

\[
    u_i(x_i, t) = \sum_m a_i^m(t)N^m(x_i), \quad (7)
\]

\[
    \phi_i(x_i, t) = \sum_n c_i^n(t)N^n(x_i). \quad (8)
\]

After some algebra, the equation for degree of freedom \( i \) of vertex \( n \) is

\[
    -\int_V \sigma_{ij}N_j^i \, dV + \int_{S_T} T_iN^n \, dS + \int_V f_iN^n \, dV - \int_V \rho \sum_m \ddot{a}_i^mN^mN^n \, dV = 0 \quad (9)
\]
Using numerical quadrature we convert the integrals to sums over the cells and quadrature points

\[ - \sum_{\text{vol cells quad pts}} \sum_{\text{quad pts}} \sigma_{ij} N^n_{ij} w_q |J_{\text{cell}}| + \sum_{\text{surf cells quad pts}} \sum_{\text{quad pts}} T_i N^n w_q |J_{\text{cell}}| \]

\[ + \sum_{\text{vol cells quad pts}} \sum_{\text{quad pts}} f_i N^n w_q |J_{\text{cell}}| \]

\[ - \sum_{\text{vol cells quad pts}} \sum_{\text{quad pts}} \rho \sum_m \ddot{a}_i^m N^m N^n w_q |J_{\text{cell}}| = \vec{0} \] (10)
Quasistatic Solution
Neglect inertial terms

Form system of algebraic equations

\[ A(t)\vec{u}(t) = \vec{b}(t) \]  \hspace{1cm} (11)

where

\[ A_{ij}^{nm}(t) = \sum_{\text{vol cells quad pts}} \sum \frac{1}{4} C_{ijkl}(t)(N_i^m + N_k^m)(N_j^n + N_i^n)w_q|J_{\text{cell}}| \]  \hspace{1cm} (12)

\[ b_i(t) = \sum_{\text{surf cells quad pts}} \sum T_i(t)N^n w_q|J_{\text{cell}}| + \sum_{\text{vol cells quad pts}} \sum f_i(t)N^n w_q|J_{\text{cell}}| \]  \hspace{1cm} (13)

and solve for \( \vec{u}(t) \).
Implementation: Fault Interfaces

Use cohesive cells to control fault behavior

Original Mesh

Mesh with Cohesive Cell

PyLith Fault Implementation
Fault Slip Implementation

Use Lagrange multipliers to specify slip

- System without cohesive cells
  - Conventional finite-element elasticity formulation
    \[
    A\ddot{u} = \bar{b}
    \]
  - Fault slip associated with relative displacements across fault
    \[
    C\ddot{u} = \bar{d}
    \]

- System with cohesive cells
  \[
  \begin{pmatrix}
  A & C^T \\
  C & 0
  \end{pmatrix}
  \begin{pmatrix}
  \ddot{u} \\
  \bar{l}
  \end{pmatrix}
  =
  \begin{pmatrix}
  \bar{b} \\
  \bar{d}
  \end{pmatrix}
  \]

- Lagrange multipliers are tractions associated with fault slip
- Prescribed (kinematic) slip
  Specify fault slip (\(\bar{d}\)) and solve for Lagrange multipliers (\(\bar{l}\))
- Spontaneous (dynamic) slip
  Adjust fault slip to be compatible with fault constitutive model
Implementing Fault Slip with Lagrange multipliers

- **Advantages**
  - Fault implementation is local to cohesive cell
  - Solution includes forces generating slip (Lagrange multipliers)
  - Retains block structure of matrix, including symmetry
  - Offsets in mesh mimic slip on natural faults

- **Disadvantages**
  - Cohesive cells require adjusting topology of finite-element mesh
Ingredients for Running PyLith

- Simulation parameters
- Finite-element mesh
  - Mesh exported from LaGriT
  - Mesh exported from CUBIT
  - Mesh constructed by hand (PyLith mesh ASCII format)
- Spatial databases for physical properties, boundary conditions, and rupture parameters
  - SCEC CVM-H or USGS Bay Area Velocity model
  - Simple ASCII files
Spatial Databases
User-specified field/value in space

- **Examples**
  - Uniform value for Dirichlet (0-D)
  - Piecewise linear variation in tractions for Neumann BC (1-D)
  - SCEC CVM-H seismic velocity model (3-D)

- Generally independent of discretization for problem

- **Available spatial databases**
  - UniformDB  Optimized for uniform value
  - SimpleDB  Simple ASCII files (0-D, 1-D, 2-D, or 3-D)
  - SCECCVMH  SCEC CVM-H seismic velocity model v5.3
  - ZeroDispDB  Special case of UniformDB
Features in PyLith 1.5
Enhancements and new features in blue

- Time integration schemes and elasticity formulations
  - Implicit for quasistatic problems (neglect inertial terms)
    - Infinitesimal strains
    - Small strains
  - Explicit for dynamic problems
    - Infinitesimal strains with sparse system Jacobian
    - Infinitesimal strains with lumped system Jacobian
    - Small strains with sparse system Jacobian

- Bulk constitutive models
  - Elastic model (1-D, 2-D, and 3-D)
  - Linear and Generalized Maxwell viscoelastic models (3-D)
  - Power-law viscoelastic model (3-D)
  - Linear Maxwell viscoelastic model (2-D)
  - Drucker-Prager elastoplastic model (3-D)
Boundary and interface conditions
- Time-dependent Dirichlet boundary conditions
- Time-dependent Neumann (traction) boundary conditions
- Absorbing boundary conditions
- Kinematic (prescribed slip) fault interfaces w/multiple ruptures
- Dynamic (friction) fault interfaces
- Time-dependent point forces
- Gravitational body forces

Fault constitutive models
- Static friction
- Linear slip-weakening
- Dieterich-Ruina rate and state friction w/ageing law
Features in PyLith 1.5 (cont.)
Enhancements and new features in blue

- Automatic and user-controlled time stepping
- Ability to specify initial stress state
- Importing meshes
  - LaGriT: GMV/Pset
  - CUBIT: Exodus II
  - ASCII: PyLith mesh ASCII format (intended for toy problems only)
- Output: VTK files
  - Solution over volume
  - Solution over surface boundary
  - State variables (e.g., stress and strain) for each material
  - Fault information (e.g., slip and tractions)
- Automatic conversion of units for all parameters
Long-term priorities
- Multi-cycle earthquake modeling
  - Resolve interseismic, coseismic, and postseismic deformation
  - Elastic/viscoelastic/plastic rheologies
  - Coseismic slip, afterslip, and creep
- Efficient computation of 3-D and 4-D Green’s functions
- Scaling to 1000 processors

Short-term priorities
- Implement several new feature and improve parallel performance
- Increase user training using virtual workshops
  - CIG/SCEC/NASA/NSF workshop: annual → biannual (Jun 2012)
  - Jun 20-24, 2011: PyLith training via virtual workshop
PyLith Development
Planned Releases

• v1.6 (June 2011)
  • HDF5 output (parallel, binary I/O)
  • Custom preconditioner with AMG solver
  • Uniform, global mesh refinement
  • Numerical damping via viscosity for dynamic problems

• v1.7 (Fall 2011)
  • Accelerate FE integrations using GPUs
  • Scalable mesh distribution among processors
  • Attenuation for dynamic simulations (wave propagation)

• v2.0 (June 2012)
  • Coupling of quasistatic and dynamic simulations
  • Heat and fluid flow coupled to elastic deformation
  • Higher order FE basis functions
  • Moment tensor point sources
  • Support for incompressible elasticity
Design Philosophy
Modular, extensible, and smart

- Code should be flexible and modular
- Users should be able to add new features without modifying code, for example:
  - Boundary conditions
  - Bulk constitutive models
  - Fault constitutive models
- Input/output should be user-friendly
- Top-level code written in Python (expressive, dynamic typing)
- Low-level code written in C++ (modular, fast)
PyLith Design: Focus on Geodynamics

Leverage packages developed by computational scientists

PyLith

- Pyre
- Sieve
- Proj.4
- FIAT

- PETSc
- numpy

- MPI
- BLAS/LAPACK
- boost
PyLith as a Hierarchy of Components

Components are the basic building blocks
PyLith as a Hierarchy of Components

PyLith Application and Time-Dependent Problem

**PyLithApp**

- **properties**:
  - none

- **facilities**:
  - mesh_generator
  - problem
  - petsc

**TimeDependent**

- **properties**:
  - dimension

- **facilities**:
  - normalizer
  - materials
  - bc
  - interfaces
  - gravity_field
  - formulation
PyLith as a Hierarchy of Components
Fault with kinematic (prescribed slip) earthquake rupture

**FaultCohesiveKin**

- **properties**
  - id
  - name
  - up_dir
  - normal_dir

- **facilities**
  - quadrature
  - eq_srcs
  - output

**EqKinSrc**

- **properties**
  - origin_time

- **facilities**
  - slip_function
PyLith as a Hierarchy of Components
Diagram of simple toy problem
PyLith as a Hierarchy of Components

[Diagram showing a hierarchy of components in PyLith, including classes like PyLithApp, MeshImporter, MeshIOCubit, TimeDependent, FacilityArray, NondimElasticQuasistatic, DirichletBC, SimpleDB, SolverLinear, TimeStepUniform, OutputSoln, OutputSolnSubset, ElasticIsotropic3D, and their relationships through methods and attributes.]
PyLith Application Flow

**PyLithApp**

main()
  - mesher.create()
  - problem.initialize()
  - problem.run()

**TimeDependent (Problem)**

initialize()
  - formulation.initialize()

run()
  - while (t < tEnd)
    - dt = formulation.dt()
    - formulation.prestep(dt)
    - formulation.step(dt)
    - formulation.poststep(dt)

**Implicit (Formulation)**

initialize()

prestep()
  - set values of constraints

step()
  - compute residual
  - solve for disp. incr.

poststep()
  - update disp. field
  - write output
Unit and Regression Testing
Automatically run more than 1800 tests on multiple platforms whenever code is checked into the source repository.

- Create tests for nearly every function in code during development
  - Remove most bugs during initial implementation
  - Isolate and expose bugs at origin
- Create new tests to expose reported bugs
  - Prevent bugs from reoccurring
- Rerun tests whenever code is changed
  - Code continually improves (permits optimization with quality control)
- Binary packages generated automatically upon successful completion of tests
- Additional full-scale tests are run before releases
Example of Automated Building and Testing

Test written to expose bug, buildbot shows tests fail.
Example of Automated Building and Testing

Bug is fixed, buildbot shows tests pass

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:21:56</td>
<td></td>
</tr>
<tr>
<td>12:23:37</td>
<td></td>
</tr>
<tr>
<td>12:24:38</td>
<td></td>
</tr>
<tr>
<td>12:25:03</td>
<td></td>
</tr>
<tr>
<td>12:25:38</td>
<td></td>
</tr>
<tr>
<td>12:26:03</td>
<td></td>
</tr>
<tr>
<td>12:26:44</td>
<td></td>
</tr>
<tr>
<td>12:30:04</td>
<td></td>
</tr>
<tr>
<td>12:30:46</td>
<td></td>
</tr>
<tr>
<td>12:32:00</td>
<td></td>
</tr>
<tr>
<td>12:32:22</td>
<td></td>
</tr>
<tr>
<td>12:41:55</td>
<td></td>
</tr>
</tbody>
</table>

Compiling/linking problem w/Darwin.

- default tests stdio
- default installation stdio
- default compile stdio
- default tests failed stdio
- default installation stdio
- default compile stdio
- default tests stdio
- binaries shipping stdio
- binaries packaging stdio
- binaries tests stdio
- binaries stdio
- binaries compile stdio

PyLith Testing