# Crustal Deformation Modeling Tutorial

## Running PyLith in Parallel

Brad Aagaard
Matthew Knepley
Charles Williams

COMPUTATIONAL INFRASTRUCTURE for GEODYNAMICS

June 28, 2013

# Concepts Covered in this Session

- Running PyLith in parallel
  - Desktop or laptop with multiple processors and/or cores
  - Cluster with multiple compute nodes
- Optimizing PyLith performance
- Building PyLith from source
- Using PyLith with a queue system
- PyLith parallel performance

# Running PyLith in Parallel

- Laptop/Desktop
  - Reduce runtime (distribute floating point operations)
  - Available with binary `--nodes=NUMCORES`
- Cluster
  - Reduce runtime (distribute floating point operations)
  - Run larger problems (distribute memory usage)
  - Must build from source for proper configuration
  - Requires additional parameters for batch submission

# How PyLith Runs in Parallel

## Single Program, Multiple Data (SPMD) Parallel Processing

1. Process 0
   1. Read in mesh
   2. Add cohesive cells by adjusting the mesh topology
   3. Partition mesh and determine vertices shared by multiple processes
   4. Distribute relevant portion of mesh to each process
2. All processes
   1. OPTIONAL Each processor refines its part of the mesh
   2. Each processor solves equations on its portion of the mesh, exchanging information as necessary with its neighbors

# Optimizing PyLith Performance

## Efficiency depends on choice of parameters and hardware

- Output

  **VTK** Inefficient: each process sends it data to process 0 for writing

  **HDF5** Efficient: each process writes its own data in binary

- Solver performance

  **Quasi-static** Field split with algebraic multigrid generally scales better than Additive Schwarz

  **Dynamic** Trivial solve scales extremely well

- Overall performance
  - Speed of memory and connection b/t memory and CPU is more important than CPU speed
  - Marginal speed improvement if you compile source for your hardware

# Running PyLith in Parallel on a Desktop

## Reduce runtime using multiple processors and/or cores

Add number of processes (usually number of cores) as argument:

```
pylith --nodes=NUMCORES
```

- PyLith binary
  - Allows interprocess communication only within a single computer
  - Works with laptops, desktops, and a single compute node
- Building PyLith from source
  - Permits optimizing code for your hardware; may provide modest performance gains

# Running PyLith on a Cluster

## For large quasi-static and 3-D dynamic simulations

- Differences from usage on a desktop machine
  - Usually requires additional parameters for batch queue system
  - **Strongly** recommend using `DataWriterHDF5Ext*` for output
    - Parallel output via shared or parallel file system
    - More failsafe than regular HDF5 output (`DataWriterHDF5*`)
  - Stdout and stderr are written to a log file or files
- Consult your system administrator on MPI parameters
- Read getting started guides provided by computing centers
  - What compiler suites and MPI versions are available?
  - What filesystems are available? Which support parallel I/O?

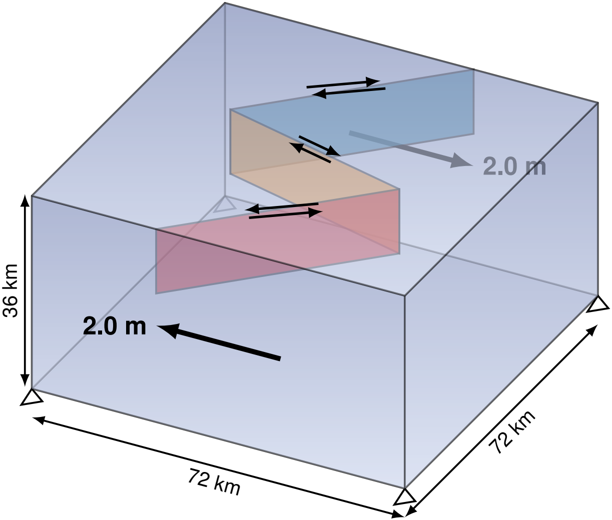
# Hints for Running PyLith on a Cluster

## How many compute nodes and cores should I use?

- General
  - If you have N compute nodes and want to run J jobs, use N/J compute nodes
  - Use the maximum number allowable by the queue with the shortest wait
  - Don't overload a compute node (memory use exceeds that available)
  - Don't overload fileservers
- Quasi-static problems
  - Memory use and runtime depends on the solver parameters
  - Memory use is often dominated by the sparse matrix
  - Different bulk rheologies use different amounts of memory
- Dynamic problems
  - Don't overload compute nodes
  - Spontaneous rupture uses more memory than prescribed slip

# PyLith Parallel Performance Test

Static solution of prescribed slip on multiple faults



# PyLith Parallel Performance Test

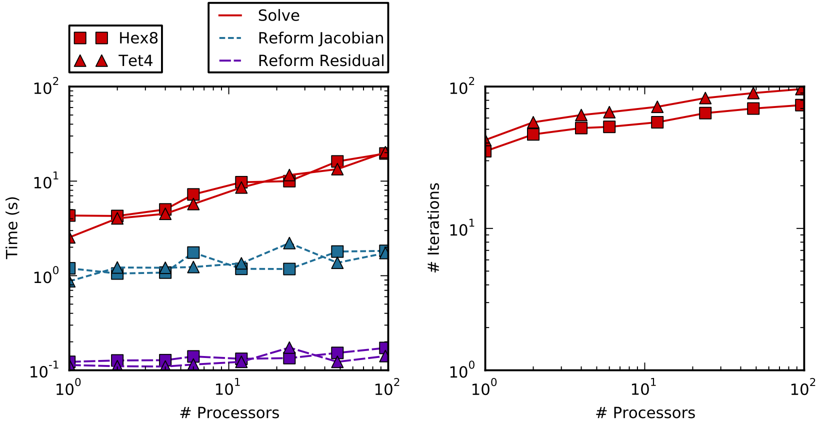
## Field split and AMG with custom fault preconditioner performs best

Number of Iterations in Linear Solve

| Preconditioner | Cell | Problem Size S1 | S2 | S4 |
|---|---|---|---|---|
| ASM | Tet4 | 184 | 217 | 270 |
| | Hex8 | 143 | 179 | 221 |
| Schur (full) | Tet4 | 82 | 84 | 109 |
| | Hex8 | 54 | 60 | 61 |
| Schur (upper) | Tet4 | 79 | 78 | 87 |
| | Hex8 | 53 | 59 | 57 |
| FieldSplit (add) | Tet4 | 241 | 587 | 585 |
| | Hex8 | 159 | 193 | 192 |
| FieldSplit (mult) | Tet4 | 284 | 324 | 383 |
| | Hex8 | 165 | 177 | 194 |
| FieldSplit (mult,custom) | Tet4 | 42 | 48 | 51 |
| | Hex8 | 35 | 39 | 43 |

# PyLith Parallel Performance Test

## Weak scaling of field split w/AMG and custom fault preconditioner



# Dissecting PETSc Log Summary I

## Where are the bottlenecks?

### Quasi-Static Simulation Savage-Prescott Benchmark

```
Summary of Stages:   ----- Time ------
                       Avg     %Total
 0:      Main Stage: 5.7474e+00   0.2%
 1:         Meshing: 1.6935e+01   0.6%
 2:           Setup: 1.4908e+01   0.5%
 3: Reform Jacobian: 5.2058e+00   0.2%
 4: Reform Residual: 1.4038e+02   5.1%
 5:           Solve: 2.3698e+03  85.8%
 6:         Prestep: 1.2892e+01   0.5%
 7:            Step: 6.8484e+01   2.5%
 8:        Poststep: 1.2611e+02   4.6%
 9:        Finalize: 4.6684e-01   0.0%
```

### Dynamic Simulation SCEC Dynamic Rupture Benchmark TPV102

```
Summary of Stages:   ----- Time ------
                       Avg     %Total
 0:      Main Stage: 3.4975e+00   0.1%
 1:         Meshing: 1.5496e+02   3.6%
 2:           Setup: 8.6702e+01   2.0%
 3: Reform Jacobian: 3.5730e+00   0.1%
 4: Reform Residual: 2.5464e+03  59.6%

 6:         Prestep: 3.1002e+00   0.1%
 7:            Step: 1.2559e+03  29.4%
 8:        Poststep: 2.1854e+02   5.1%
 9:        Finalize: 1.0401e+00   0.0%
```

# Dissecting PETSc Log Summary II

## Identify memory bandwidth saturation and communication bottlenecks

| Event | # Cores | Load Imbalance | MFlops/s | Comments |
|---|---|---|---|---|
| VecMDot | 1 | 1.0 | 2007 | |
| | 2 | 1.1 | 3809 | |
| | 4 | 1.1 | 5431 | |
| | 6 | 1.1 | 5967 | Memory bandwidth saturation |
| | 12 | 1.2 | 5714 | |
| | 24 | 1.2 | 11784 | Multiple compute nodes, scaling returns |
| | 48 | 1.2 | 20958 | |
| | 96 | 1.3 | 17976 | Inter-compute node saturation? |
| VecAXPY | 1 | 1.0 | 1629 | |
| | 2 | 1.1 | 3694 | |
| | 4 | 1.1 | 5969 | |
| | 6 | 1.1 | 6028 | Memory bandwidth saturation |
| | 12 | 1.2 | 5055 | |
| | 24 | 1.2 | 10071 | Multiple compute nodes, scaling returns |
| | 48 | 1.2 | 18761 | |
| | 96 | 1.3 | 33676 | |
| VecMAXPY | 1 | 1.0 | 1819 | |
| | 2 | 1.1 | 3415 | |
| | 4 | 1.1 | 5200 | |
| | 6 | 1.1 | 5860 | Memory bandwidth saturation |
| | 12 | 1.2 | 6051 | |
| | 24 | 1.2 | 12063 | Multiple compute nodes, scaling returns |
| | 48 | 1.2 | 23072 | |
| | 96 | 1.3 | 28461 | |

# PyLith v1.9 versus v2.0

## Under-the-hood improvements fix some parallel scaling issues

- PyLith v1.9
  - Reading in mesh by single process limits size of calculation
  - Dynamic problems with $>$50M cells (hex or tet)
  - Memory imbalance of up to 10x for large problems with faults
- PyLith v2.0
  - Reading in mesh by single process limits size of calculation
  - Improved mesh data structures reduce memory use
  - Expect memory balancing to be very good
  - Expect to be able to run problems with $O(10^8)$ cells

# Building PyLith from Source

## Required for PyLith to use multiple compute nodes on a cluster

**Use the PyLith installer utility!!!**

`pylith-installer-1.9.0-0.tgz`

- Downloads, configures, and builds PyLith and dependencies
- User selects which dependencies are needed and installer will do some minimal checks
- Insures versions, configuration, and builds are consistent PyLith requirements
- See INSTALL file in installer tarball for instructions

# Submitting Jobs to PBS Queue System

## PBS is one of the most common batch queue systems

- PyLith uses Pyre to submit jobs directly to PBS
  1. Perform minimal validation of the simulation parameters
  2. Create a shell script to submit job
  3. Submit job
- Assumes you have already setup running jobs on the cluster

# Submitting Jobs to PBS Queue System

Put parameters common to all jobs in
`$HOME/.pyre/pylithapp/pylithapp.cfg`

```
[pylithapp]
scheduler = pbs

[pylithapp.pbs]
# Shell used for job script submitted to batch system
shell = /bin/bash
# Command line arguments in qsub command
# -V = Use current environment variables in batch job
# -m bea = Send email when job begins, ends, or aborts
qsub-options = -V -m bea -M johndoe@university.edu

[pylithapp.launcher]
command = mpirun -np ${nodes} -machinefile ${PBS_NODEFILE}
```

# Submitting Jobs to PBS Queue System

Pass job specific parameters via the command line

`--nodes=NPROCS` Total number of processes

`--scheduler.ppn=PPN` Number of processes per compute node

`--job.name=NAME` Name of job

`--job.stdout=LOG_FILE` File where stdout is written

$$\mathtt{NPROCS} = \mathtt{NCOMPUTENODES} \times \mathtt{PPN}$$

# Debug Launching Parallel Jobs on Queue System

Use command line help features to see commands being processed

- See default and set parameters

  `--COMPONENT.help-properties` See properties and their values

  `--COMPONENT.help-components` See subcomponents

  `pylithinfo PYLITH_ARGS` Dumps all parameters to `pylith_parameters.txt`

- Submitting to the queue (scheduler)

  `--scheduler.help` See list of properties/components available

  `--scheduler.dry` Dump script for batch submission to stdout

- Launching job on compute nodes (launcher)

  `--launcher.help` Total number of processes

  `--launcher.dry` Dump launching command to stdout