Software Integration Group

Michael McLennan
Chief Software Architect
Martin Hunt
Senior Software Engineer
Lin Sun
Software Engineer
Gazi Yildirim
Software Engineer

1. Create MEMOSA integration framework
2. Work with others to integrate code
3. Set up regression testing and monitor results
## Software Components

<table>
<thead>
<tr>
<th>Code</th>
<th>From</th>
<th>Lang</th>
<th>LOC</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAMMPS</td>
<td>Sandia</td>
<td>C++</td>
<td>118,000</td>
<td>MD simulator</td>
</tr>
<tr>
<td>FVM</td>
<td>Purdue</td>
<td>C++</td>
<td>10,000</td>
<td>Core for FVM solver</td>
</tr>
<tr>
<td>MPM</td>
<td>UNM</td>
<td>F90/F77/C</td>
<td>25,000</td>
<td>Core for MPM solver</td>
</tr>
<tr>
<td>Rar Gas Dyn</td>
<td>Purdue</td>
<td>F90</td>
<td>6,000</td>
<td>Boltzmann-ESBGK</td>
</tr>
<tr>
<td>Reactive MD</td>
<td>Purdue</td>
<td>C</td>
<td>27,500</td>
<td>Reactive force fields</td>
</tr>
<tr>
<td>Dislocations</td>
<td>Purdue</td>
<td>C/F77</td>
<td>2,000</td>
<td>Micromechanical models</td>
</tr>
<tr>
<td>Rappture</td>
<td>Purdue</td>
<td>C/C++/XML</td>
<td>60,000</td>
<td>Input/output handling</td>
</tr>
<tr>
<td>MEMOSA</td>
<td>Purdue</td>
<td>C/C++/Python</td>
<td>10,000</td>
<td>Simulation framework</td>
</tr>
<tr>
<td>Im Boundary</td>
<td>Purdue</td>
<td>C</td>
<td>5,000</td>
<td>FVM / MPM connection</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>Purdue</td>
<td>C++</td>
<td>10,000</td>
<td>Sensitivity analysis</td>
</tr>
<tr>
<td>UQ drivers</td>
<td>Purdue</td>
<td>C/Python</td>
<td>5,000</td>
<td>Uncertainty quantification</td>
</tr>
<tr>
<td>Models</td>
<td>Purdue</td>
<td>C/F77</td>
<td>5,000</td>
<td>Various physical models</td>
</tr>
</tbody>
</table>
MEMOSA Architecture
Object-Oriented Infrastructure

- C++/F90 core for solvers
- Python objects for high-level scripting
- Shared object plug-in architecture
- C++ templates for arithmetic types
Sensitivity Framework*

C++/F90 code overloads operators to compute functions + derivatives

Template <class T>
void myfunc (  
    const T& x, const T& y, // inputs
    T& p, T& q          // outputs
) {
    p=3*x*x+sin(y);
    q=p/y;
}

myfunc<double>(x,y,p,q);   usual function
myfunc<Tangent>(x,y,p,q);  values + derivatives

\[
\begin{align*}
    p &= 3x^2 + \sin(y) \\
    p' &= 3(x^2 + x'x) + \cos(y)y'
\end{align*}
\]

* Only for device-level simulations—not LAMMPS/micromechanical models
Generating Python Bindings

SWIG - http://www.swig.org

Builds Python bindings automatically based on interface desc in .i files

\[
fmodel = \text{models}.\text{FlowModelA}(\text{geomFields, flowFields})
\]

\[
fmodel.\text{init}()
\]

\[
fmodel.\text{advance}(\text{numIterations})
\]

\[
\text{template<class T>}
\]

\[
\text{class } \text{FlowModel} : \text{public Model} \{
\text{public:}
\]

\[
\text{FlowModel}(\text{const GeomFields}& \text{geomFields},
\]

\[
\text{virtual } \sim\text{FlowModel}();
\]

\[
\text{virtual void } \text{init}();
\]

\[
\text{void } \text{advance}(\text{const int niter});
\]
Regression Testing

System Tests

Unit Tests

… dozens of end-to-end tests

… hundreds of small tests

```python
foptions = fmodel.getOptions()
foptions.momentumTolerance=1e-3
foptions.continuityTolerance=1e-3
foptions.setVar("momentumURF",0.7)
foptions.setVar("pressureURF",0.3)
...
```
Uncertainty Quantification

Stochastic Collocation Methods

Choice of inputs from collocation

Solver

realizations

Python scripts drive collocation runs

Stochastic Collocation

+ Easier to implement
+ Covers everything—solver, LAMMPS, etc.
+ Should work fine for number of inputs

Stochastic Galerkin Methods
Compact Models

C/C++ Data Objects

Snippets of source code

Parameters & Lookup Tables & Constitutive Laws

Results & Metadata

MD/micromechanical tools:

- Build a collection of results
  - Results
  - Metadata to reproduce
- Distill down to compact models
- Models are called for evaluation in FVM/MPM code
Parallelization of Solvers

MEMOSA library manages point-to-point communication and data exchange between MPM/FVM solvers.

- PETSc / ParMETIS for domain decomposition
- Iterative solvers & pre-conditioners: SPIKE, PETSc
- Use Vtune / Tau / PAPI for optimization
Hardware Plans

Steele
848 x 8 core Dell 1950
60 teraFLOPS

100 node cluster in Spring
Matching funds from Purdue

LLNL unclassified Purple

LLNL ALC
Verification & Validation

- Data captured in Subversion / Wiki
- Data from experiments feeds modeling efforts
- Regression tests run regularly
- Comparison with experiments
Project Management

https://developer.nanohub.org/projects/app-memosa

• Integrated Subversion source code control system
• Wiki for software documentation and project notes
• Bug tracking system
Release Plans

- Release stable versions starting January 2010
- Open source license for core solvers
- Documentation: Doxygen + Wiki notes
- Offer downloads through nanoHUB.org