Power uprates and plant life extension

Engineering design and analysis

Science-enabling high performance computing

Fundamental science

Plant operational data

L3:VRI.PSS.P1.01
Rod Schmidt
SNL
Completed: 10/29/10
Introduction to LIME: A Lightweight Integrating Multi-physics Environment for Coupling Codes

Rod Schmidt
Sandia National Labs

LIME version 0.6
Oct 29, 2010
Outline

• Some Multi-physics Coupling Issues and Terms

• Background on LIME Development

• Description of LIME

• Use of LIME to create multi-physics applications
What do we mean by “coupled” physics?

• The solution of one physics equation (set) is dependent upon the solution of another different physics equation (set).

• Physics coupling can be
  – “one-way” or “two-way”
  – “linear” or “non-linear”
  – “strong” or “weak”

• Physics coupling may occur
  – Within a shared spatial domain
  – Across an interface
  – Through action at a distance

• For our purposes, “multi-scale” coupling and “multi-fidelity” coupling will simply be considered as particular types of coupled systems.
“Physics coupling” and a “numerical solution strategy” are distinctly different things.

• “Physics coupling” relates to the actual physics
  – “strong” or “weak”
  – “one-way” or “two-way”
  – “linear” or “non-linear”, etc.

• “Numerical solution strategy” relates to the approach and approximations made to obtain numerical solutions to the coupled physics equations.
  – “loosely” coupled transient solution method
    * Explicit, time-lagged, . .
    * Operator split methods, . . .
    * Implicit, but solved using Picard iteration (sometimes called weak coupling)
  – “tightly” coupled transient solution method
    * Implicit
    * Fully coupled non-linear solution methods such as Newton, or JFNK (sometimes called strong coupling in this context)

• Because similar terms are sometimes used when discussing these different topics, care must be taken to avoid confusion.
The need to address complex coupled non-linear multi-physics problems is not new.

- Scientists and engineers in the NE community (among many others) have wrestled with these types of problems for years
  - Remarkably useful computer codes have been developed and successfully applied, despite their limitations.
  - Many of these codes remain, even today, as “state-of-the-art” codes that reflect many years of R&D investment.

- However, computational tools, capabilities, and resources have changed dramatically, and will continue to do so.

- How to “best” solve these types of problems remains an area of active research, with many facets.

- LIME, initially developed as part of an LDRD, represents the early stage of one approach whose characteristics were driven by a particular set of needs and desired capabilities.
What is LIME?

LIME is:

• an acronym for Lightweight Integrating Multi-physics Environment for coupling codes.

• a tool for creating multi-physics simulation code(s) that is particularly useful when computer codes are currently available to solve different parts of a multi-physics problem.

• intended to provide
  – key high-level software,
  – a well defined approach (including example templates),
  – and interface requirements for participating physics codes to enable the assembly of these codes into a robust and efficient multi-physics simulation capability.

• one part of the larger VERA framework being developed in CASL.
Important Characteristics of LIME

• **Designed to:**
  - Work with advanced solver frameworks (e.g. as extensions to the Trilinos/NOX nonlinear solver library)
  - Enable separate physics codes (“new” and “old”) to be combined into a robust and efficient fully-coupled multi-physics simulation capability
  - Minimize the requirements barrier for an code application to participate without limiting the sophistication of the code applications.
  - Preserve and leverage any specialized algorithms and/or functionality an application may provide.
  - Allow composition of both controlled and open-source components, enabling protection of export-controlled or proprietary code while still allowing distribution of the core system and open components.

• **Is not limited to:**
  - Codes written in one particular language
  - A particular numerical discretization approach (e.g. Finite Element)
  - Physical models expressed as PDE’s.

• **Is not “plug and play”:**
  - Requires revisions/modifications to most stand-alone physics codes
  - Requires the creation of customized “model evaluators”
Why is LIME called “lightweight?”

Because of two design objectives:

(1)
• to keep the main software relatively small in size and complexity
• to require only a few standard libraries to build (all openly available)
• to be easily portable to a wide range of computing platforms.

(2)
• to minimize the constraints placed on the codes and models to be incorporated

Note: A tradeoff closely linked to these objectives is that to create a new multi-physics application using LIME, some amount of customized software must be written.
Key components of a simple generic application created using LIME

- **Base LIME software**

- **Level 1**
  - Input File A
  - Physics A
  - Model Evaluator A

- **Level 2**
  - Input File B
  - Physics B
  - Model Evaluator B

- **Level 3**
  - Input File C
  - Physics C
  - Model Evaluator C

- **Multi-Physics Driver**

- **Problem Manager**

- **PM Input Files (xml)**

- **Trilinos, NOX Solver Library**
Key Tasks of the **Multi-Physics Driver**

1. Do all set-up tasks for the problem that is to be run, including creating a Problem Manager specific to the problem. The set-up phase includes the following specific tasks:
   * If running in parallel, create the mpi communicator based on the number of processors requested.
   * Read the XML input file
   * Create the multi-physics coupling Problem_Manager object with solution strategy and MPI communicator
   * Initialize transfer object creation
   * Set up (create) and register the physics package objects for each Model Evaluator / Physics Code pair.
   * Set up data transfers between Model Evaluators
   * Call the Problem_Manager to set up the output

2. Call the Problem Manager to solve the problem

3. Gracefully end the simulation
Key Tasks of the Problem Manager

1. Create the global state vector $X$, and define the physics-specific state vectors $(X_A, X_B, X_C, \ldots)$

2. Perform time integration, including the following for each time step:
   - Perform time-step control: Negotiate/calculate based on all the physics.
   - Request a "predicted" solution state for time step $n+1$ from each physics code (if available) through its model evaluator.
     * Obtain a converged solution for time step $n+1$ using solution strategy defined by the user input. Current options include fixed point iteration and JFNK using the Trilinos/NOX nonlinear solver library.
       o Request residuals from physics codes
       o Request physics-based preconditioning
       o Update state vector
   - Perform output (each physics code)
The role of customized “model evaluators”

1. A customized model evaluator (ME) must be written for each physics code being coupled into the multi-physics application.

2. Each ME inherits the Problem Manager (PM) base class and implements supported PM interfaces to the underlying physics code.

3. Each ME has control of one defined piece of the overall state vector.

4. Each ME has access to all other state vector pieces that appear directly or indirectly (e.g. through property variation affects) in its equation set or models.

5. Model Evaluators communicate up to the Problem Manager, between themselves, and down to a specific Physics Code.
Revisions and Modifications that may be required of a **Physics-Code**

- Console IO must be redirected (no pause statements or read/write to standard streams)

- Each code must be wrapped so the multi-physics driver can link to it (i.e. like a library).

- Each code must be organized into several key parts that can be called independently.
  - Initialization: *read inputs, allocate memory...*
  - Solve: *compute solution for a given time step and “state”*
  - Advance: *copy converged “state” and prepare for next step*
  - Output: *print to output files*

- Additional routines that may be needed:
  - Register coupling capabilities
  - Pass control variables
  - Compute and pass data for coupling to other physics
  - Compute residuals
  - Perform preconditioning
LIME has been used to assemble several different multi-physics applications

- LAAMPS - SPARKS coupling
- Stress corrosion cracking (4 codes)
- Development of a proto-type systems-level SFR safety analysis code at Sandia National Labs (BRISC)
Some example modifications and revisions to RIO (used for BRISC) that were made to interface with LIME

- Revised high-level code structure. For example, routines called in “main” went from

```
parallel_init()
parallel_sync()
init_mem()
get_parallel_filename()
read_exodus_mesh_file()
read_input_file()
initialize_plot_curves();
initialize_linear_solver()
initialize_assemble()
set_initial_conditions()
restart_solution()
initialize_results_file()
rehash_input()
integrate()
global_rmax()
```

- And in “integrate”,

```
before_time_integration
- actions needed before taking one time step
time_integration
- integrate over one time step
after_time_integration
- actions needed after taking one time step
select_time_step
- compute time step value
```
Some example modifications and revisions to RIO (used for BRISC) that were made to interface with LIME (continued)

- Wrapped as a C++ library

- Created several new routines and some new data structures (arrays) for various purposes. For example

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonlinear_resid</td>
<td>Compute total nonlinear residual vector</td>
</tr>
<tr>
<td>compute_precond_matrix</td>
<td>Compute preconditioning matrix for jfnk</td>
</tr>
<tr>
<td>precondition_for_jfnk</td>
<td>Perform preconditioning for jfnk</td>
</tr>
<tr>
<td>save_state</td>
<td>Save current state before precondition. solve</td>
</tr>
<tr>
<td>reset_state</td>
<td>Reset current state with X_TOT values</td>
</tr>
<tr>
<td>dx_sol_reset_state</td>
<td>Compute precondition dir. vector and reset</td>
</tr>
</tbody>
</table>

- Modified each of the solve routines to facilitate storing new quantities needed
Some high-level basic steps to creating a new multi-physics application using LIME (1)

Part 1

• Define the target problem (or class of problems) and identify the different codes to be coupled
• Fully describe the coupled physics problem of interest in terms of
  – the basic equations being solved by the different physics codes and associated local state variables
    * the type and nature of the coupling
    * shared spatial domain, across an interface, . . .
    * one-way or two-way
    * strong or weak
  – The coupling terms and transfer of information required
    * Will special transfer operations need to be created?
    * Will non-linear ilimation be required?
  – The global set of state variables to be evolved by the Multi-Physics Driver
Some high-level basic steps to creating a new multi-physics application using LIME (continued)

Part 2

• Construct a feasible code coupling map-diagram
  – Include each “physics code <-> model evaluator” pair
  – Define all communication/data transfer needs and dependencies
    * Show special data transfer operations required
  – Look for numerical compatibility issues
    * Consistent spatial and temporal discretization schemes?
    * Can a residual be properly defined?
    * Will non-linear elimination be required?
    * Are there state-variable scaling issues?
    * Are material property models consistent?
Some high-level basic steps to creating a new multi-physics application using LIME (continued)

Part 3 (For each physics code)

• Define the Code modifications/additions needed
  – For stand-alone
  – For coupled setting

• Make changes needed to wrap the code for stand-alone situation
  – Create base model evaluator
  – Test against reference code results

• Make changes needed to couple with just one other code
  – Modify base model evaluator to address coupling needs
  – Test against a series of very simple tests

• Continue adding each coupled code, one at a time . .
In CASL, LIME sits within the larger VERA environment

- Examples of software tools that sit outside of LIME but within VERA.
  - Solver libraries (e.g. Trilinos/NOX)
  - Meshing tools (e.g. CUBIT)
  - UQ toolkit (e.g. Dakota)
  - Viz. tools (e.g. paraview, Visit, )
  - Parallel decomposition tools (e.g. Zoltan)
  - Workflow environment tools