A CASL Multiphysics Code Coupling Primer: Software Integration 101 (SAND2013-5908C)

Roger Pawlowski
Sandia National Laboratory

July 9, 2013
A CASL Multiphysics Code Coupling Primer: Software Integration 101

Roger Pawlowski
Sandia National Laboratories

Stuart Slattery and Paul Wilson
University of Wisconsin

Roscoe Bartlett
Oak Ridge National Laboratory

CASL 2013 Summer Student Workshop
Tuesday, July 8th, 2013
Motivation: What makes Multiphysics/Code Coupling Difficult?

• The most complex software engineering project I have been involved with
  – Fortran, C, C++, Java, Python, Perl, …
  – 21 git repositories
  – VERA is composed of 350+ software engineering packages, 12 TPLs

• Multiscale physics: Thermal hydraulics (CFD, Subchannel), Neutron transport (SN, MOC), materials models, crack propagation, multiphase boiling, …

• Multiple discretizations and solution algorithms
  – Steady-state, transient (explicit, operator split, implicit), pseudo-transient, continuation, eigensolvers, etc…
  – CVFEM, FE, DGFEM, DAE network models, …
  – Stability and Conservation can be critical

• Code use different units, coordinate systems, dimensions, pin axis alignment

• Software engineering quality of individual codes: app → library = disaster!

Code integrations require a strong combination of skills in physics simulation, numerical algorithms and software engineering
Tools for Multiphysics Simulation
(Spanning Individual Applications and Coupled Systems)

• Motivating Example
• A Common Language: Domain Model
• Software Engineering Framework
• Code-to-code Data Transfer Utilities
• Abstraction Layer for Abstract Numerical Algorithms
• Nonlinear Solution Algorithms
• Linear Algebra and Linear Solution Algorithms

This talk
CASL Goal: Coupling “Established” Codes to Produce A Novel Capability

- **CTF**: Fluid flow and heat transfer around fuel pins
- **Insilico**: Neutronics (energy source generation in entire reactor medium)
- **Peregrine/MOOSE**: Thermal conductivity accounting for crack propagation in cycle
A Domain Model

\[ f(\dot{x}, x, \{p_l\}, t) = 0 \]

- **Residual**
- **Time**
- **State (DOF)**
- **Set of parameters**

\[ \dot{x} = \frac{\partial x}{\partial t} \]

\[ g_j(\dot{x}, x, \{p_l\}, t) = 0, \text{ for } j = 0, \ldots, N_g - 1 \]

\[ g_j(\dot{x}, x, \{p_l\}, t) : \mathbb{R}^{2n_x + (\sum_{i=0}^{N_p-1} n_{p_i}) + 1} \rightarrow \mathbb{R}^{n_{g_j}} \] is the \( j \)th response function.

- **Input Arguments:** state time derivative, state, parameters, time
- **Output Arguments:** Residual, Jacobian, response functions, etc...

\( x \in \mathbb{R}^{n_x} \) is the vector of state variables (unknowns being solved for),
\( \dot{x} = \partial x / \partial t \in \mathbb{R}^{n_x} \) is the vector of derivatives of the state variables with respect to time,
\( \{p_l\} = \{p_0, p_1, \ldots, p_{N_p-1}\} \) is the set of \( N_p \) independent parameter sub-vectors,
\( t \in [t_0, t_f] \in \mathbb{R}^1 \) is the time ranging from initial time \( t_0 \) to final time \( t_f \),
\( f(\dot{x}, x, \{p_l\}, t) : \mathbb{R}^{2n_x + (\sum_{i=0}^{N_p-1} n_{p_i}) + 1} \rightarrow \mathbb{R}^{n_x} \)
Extension to Multiphysics

Split parameters into “coupling” and truly independent.

\[ f_i(\dot{x}_i, x_i, \{z_{i,k}\}, \{p_{i,l}\}, t) = 0 \]

- Set of coupling parameters
- Set of independent parameters

Require transfer functions:
- Can be complex nonlinear functions themselves

\[ z_{i,k} = r_{i,k}(\{x_m\}, \{p_{m,n}\}) \]

Transfer Function

Response functions now dependent on \( z \)
- Can be used as coupling parameters (\( z \)) for other codes

\[ g_{i,j}(\dot{x}_i, x_i, \{z_{i,k}\}, \{p_{i,l}\}, t) \]

Response Function
An Assortment of Coupling Algorithms

- Picard-based (Black-Box)
  - Block Nonlinear Jacobi
  - Block Nonlinear Gauss-Seidel
  - Anderson Acceleration
- Newton Based (Block Implicit)
  - Jacobian-free Newton-Krylov
  - Newton-Krylov (Explicit Jacobian)
  - Nonlinear Elimination (Schur complement formulation)

Example: Two Component system

\[
\begin{align*}
  f_0(x_0, z_{0,0}) &= 0 \\
  f_1(x_1, z_{1,0}) &= 0 \\
  z_{0,0} &= r_{0,0}(x_1) \\
  z_{1,0} &= r_{1,0}(x_0)
\end{align*}
\]

Picard Iteration: Nonlinear Block Gauss-Seidel

Require: Initial guesses \( x_0^{(0)} \) and \( x_1^{(0)} \) for \( x_0 \) and \( x_1 \):

\[
k = 0
\]

while not converged do

\[
k = k + 1
\]

Solve \( f_0(x_0^{(k)}, r_{0,0}(x_1^{(k-1)})) = 0 \) for \( x_0^{(k)} \)

Solve \( f_1(x_1^{(k)}, r_{1,0}(x_0^{(k)})) = 0 \) for \( x_1^{(k)} \)

end while

Newtont-based

\[
\begin{bmatrix}
  \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial z_{0,0}} \\
  \frac{\partial f_1}{\partial z_{1,0}} & \frac{\partial f_1}{\partial x_1}
\end{bmatrix}
\begin{bmatrix}
  \Delta x_0^{(k)} \\
  \Delta x_1^{(k)}
\end{bmatrix}
= -
\begin{bmatrix}
  f_0(x_0^{(k-1)}, r_{0,0}(x_1^{(k-1)})) \\
  f_1(x_1^{(k-1)}, r_{1,0}(x_0^{(k-1)}))
\end{bmatrix}
\]

- Off-block diagonals may be hard to compute
- Can avoid computing Jacobian by using JFNK,
- BUT you still need to precondition (\( M \approx W^{-1} \))
Picard Iteration for Solving a Nonlinear Problem

- q-linear convergence
- Self consistency of each physics
- Not globally convergent
- Contraction mapping theorem used to show when a Fixed Point Iteration will converge:
  - Requires Lipschitz continuity of fixed point map with a Lipschitz constant $\gamma < 1$
    \[ ||G(x) - G(y)|| \leq \gamma ||x - y|| \]
- Anderson Acceleration can sometimes recover stability of the method (another tool in the box)

Example: Two Component system

\[
\begin{align*}
  f_0(x_0, z_{0,0}) &= 0 \\
  f_1(x_1, z_{1,0}) &= 0 \\
  z_{0,0} &= r_{0,0}(x_1) \\
  z_{1,0} &= r_{1,0}(x_0)
\end{align*}
\]
VERA: Virtual Environment for Reactor Analysis

Baseline
- VABOC
- BOA
- ANC9
- VIPRE-W

Geometry
- MAMBA
- PEREGRINE
- Star-CCM+

Chemistry
- COBRA-TF
- Hydra-TH
- Drekar
- Denovo
- MPACT
- DeCAR

Thermo-Mechanics
- Thermal-Hydraulics
- Neutronics
- XSProc
- MOAB
- LIBMESH
- STK

Thermal-Hydraulics
- Common Input
- MOOSE
- DAKOTA
- Trilinos
- RELAP5

Geometry
- DataTransferKit

System
- LIME
- Trilinos
- DAKOTA
- MOOSE
- Common Input

Front-end
## Application Classification

Inputs and outputs are **optionally** supported by physics model → restricts allowed solution procedures

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Required Inputs</th>
<th>Required Outputs</th>
<th>Optional Outputs</th>
<th>Time Integration Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Only Model</td>
<td>$p \rightarrow g(p)$</td>
<td>$p$</td>
<td></td>
<td>$g$</td>
<td>Internal</td>
</tr>
<tr>
<td>(Coupling Elimination)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State Elimination Model</td>
<td>$p \rightarrow x(p)$</td>
<td>$p$</td>
<td></td>
<td>$x$</td>
<td>Internal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$g$</td>
<td></td>
</tr>
<tr>
<td>Fully Implicit Time Step Model</td>
<td>$f(x, p) = 0$</td>
<td>$x, p$</td>
<td></td>
<td>$f$</td>
<td>Internal</td>
</tr>
<tr>
<td>Transient Explicitly Defined ODE Model</td>
<td>$\dot{x} = f(x, p, t)$</td>
<td>$x, p, t$</td>
<td></td>
<td>$W, M, g$</td>
<td>External</td>
</tr>
<tr>
<td>Transient Fully Implicit DAE Model</td>
<td>$f(\dot{x}, x, p, t) = 0$</td>
<td>$\dot{x}, x, p, t$</td>
<td>$f$</td>
<td>$W, M, g$</td>
<td>External or Internal</td>
</tr>
</tbody>
</table>

\[ W = \alpha \frac{\partial f}{\partial \dot{x}} + \beta \frac{\partial f}{\partial x} \quad M = \text{preconditioner} \]
Peregrine/Insilico/CTF Executable
(Only ONE of many executables in VERA)

- VRIPSS
- COBRA-TF
- Exnihilio (Insilico, Denovo, nemesis)
- Drekar
- MOOSE/Peregrine
- Qt
- SCALE (200+ libraries, 30+ years of NRC codes)
- LIBMESH
- Data Transfer Kit
- LIME
- Trilinos (60+ libraries)
- PETSc
- HYPRE
- Netcdf
- HDF5
- Boost
- Many others…

We are pulling in almost every general HPC library under one executable and dealing with massive collisions!

- Library version requirements
  - Ex: 4 Apps use petsc w/ different ver/flags
  - Global variables/MACROS
The VERA Software Stack

• VERA is based on Trilinos/TriBITS build, integration and testing environment
  – Follows a TOOLKIT approach.
  – Fundamental atomic unit is a package.
  – Packages declare dependencies on other packages and TPLs
  – VERA/TriBITS provides many standard SQA practices:
    • Source code management (git).
    • Code Integration and Build tools (TriBITS/CMake-based).
    • Automated testing: Checkin, CI, Nightly, Weekly (CTest, python scripts).
    • Dashboard results and email for failing tests (CDash)

• Each VERA physics code is expected to be integrated as a “package”
  – Must convert to TriBITS/CMake
  – Can live in separate repositories (git preferred, can work with svn)

• A New package integrates existing packages
Package Dependencies Minimize Testing, Speed Integration

VRIPSS

- Drekar/Exnihilo
- Drekar Model Evaluator
- Exnihilo Model Evaluator

- LIME

Trilinos

- NOX
- ANASAZI
- 32 Trilinos packages…

- HDF5

Scale

- XSProc
- 200+ packages (30 years of codes)

Drekar/Panzer

- netcdf

Qt
Native TPL Builds **In-line with TRIBITS**
(Ext package support, Export makefiles refactored)

**Traditional Approach**

- TriBITS
  - VERA
    - DTK
      - Trilinos
  - PETSc
  - Hypre

**New Requirements**

- TriBITS
  - CASL MOOSE
    - Peregrine
    - MOOSE
      - libmesh
    - DTK
      - Trilinos
  - PETSc
  - Hypre

---

The introduction of MOOSE Created circular dependencies on VERA components

**TriBITS is now truly a meta build system!**
Testing

• Is the absolute single most important aspect of integration into VERA!

• Benefits:
  – Documents that a capability exists and does not regress
  – Documents valid use cases
  – Documents that completed milestones have not regressed
  – Allows for nearly automatic releases
Testing Requirements

• Disciplined development teams
  – Tests must pass 100% of the time
  – Broken tests must be addressed immediately or disabled and documented in backlog if a regression occurred.

• Meaningful tests!
  – Not just a running executable
    • Performance can degrade but still complete (e.g. more iterations in solvers)
    • Answers can be wrong (e.g. intel compliers for ANC/Vipre/Boa)
  – No eyeball norms!
    • Automated interrogation of results to defined tolerances, no human interaction!
  – Output must be compared against gold standard metrics
    • e.g. Solution values, quantities of interest, iteration counts, timings
TriBITS supports Multiple Test Categories

- Unit Tests (used for checkin and CI testing)
  - Short turn around that users can run before making commits to any repository.
  - Very fast tests – total suite should run in a few minutes at most
- Nightly Tests
  - Longer tests that show critical capabilities
- Weekly testing
  - Tests that take a really long time to complete
  - Usually milestones, valgrind style tests, or coverage testing
- Performance
  - Tied to a specific platform
  - Uses execution times
Need automated access to repository (ssh keys)!
Build Types

- Testing uses different combinations of compilers and build flags
  - MPI and SERIAL builds
  - DEBUG and RELEASE
  - Compilers: GCC, Intel, Hybrid
  - Explicit and Implicit Instantiation
  - Checked STL
  - Package defined ifdefs

- A combinatorial explosion – too much to test with our resources. CASL checking test script defines 2 builds.
  - MPI_DEBUG: MPI, DEBUG, Checked STL, Implicit Inst.
  - SERIAL_RELEASE: SERIAL, RELEASE Explicit inst.

- Package classification: Primary Stable, Secondary Stable, Experimental
  - Users can add extra builds for secondary stable and experimental packages

Testing requires multiple builds of the same code!
Example of Integration Workflow:
Drekar/Denovo Coupling

New mappings

Branch to test integration

Branch for new Drekar/Denovo driver

Branch to fix Denovo LIME driver for Angmar changes to Denovo

Branch for Drekar/Denovo coupling

Branch to test against current version of ExNihilo
Sample of Complex Integration Workflow: Drekar/Denovo Coupling

No way to do this efficiently without git (Dist. VC)!
Modifying a code for integration is difficult and time consuming

• Codes are no longer top of the food chain (main()):
  – No global variables, using namespace declarations in headers
  – “Solve” can be called multiple times
  – Must be able to reset if any physics fails a “step”
  – Can not control/manipulate the parse of input
  – Can not redirect output streams, must allow ostreams to be set
  – Can not assume MPI_COMM_WORLD anywhere in your code (must accept an MPI communicator)

• Exposing input parameters and responses in parallel is the most challenging aspect!
  – Parallel distribution, data structures, ...

• Memory management strategies are critical (RCPs)
• Robust error handling
The Details: Debugging Skills are Critical (CTF/Insilico/Peregrine integration specific)

- Object collisions (not namespaced) can generate seg faults
  - SCALE/MOOSE: Reporter (dtor calling wrong function)
- Multiply defined macros redefined in files
  - MOOSE/Qt
- Inconsistent compiler defines/build flags
  - Can result in mysterious seg faults from inconsistently built objects
  - PETSc/SCALE: OpenMP compilation requirements
  - Seg faults from c++11 flags injected in libmesh (no disable)
- Scoping and ownership
  - std::cout vs generic std::ostream
- Redirecting ostreems
  - grabbing rdbuf on std::cout and setting to null
  - No override of ostream
  - What about Fortran code output?
- Colliding parser/command line handling
  - Incompatible structure “--mesh=2” vs “–mesh 2“
  - Apps removing arguments from argc/argv during parse
  - Parsers throwing exceptions for invalid arguments

Debugging skills are critical for VRI team!
Units/Coordinate Complexity
(You can’t make this stuff up!)

Transfers: Must account for axis alignment, active fuel offset height, and unit conversions!
Input file consistency is critical!

- **English units**
  - Cartesian
  - Z-Axis aligned
  - No Fuel offset

- **Agn./MKS Units**
  - Cylindrical/Cart.
  - Y-Axis aligned
  - Any offset

**Coefficients for heat flux**
- Watts / (m^2 K)
- Watts / (cm^3)
- Lb_m/ft^3

**Clad surface Temperature**
- BTU / (hr ft^2 F)
- F

**Fluid Temperature**
- Fluid Density
- gm/cm^3

**Power**
- Watts / (m^3)
- K

**MOOSE**
Data Transfer Kit  
(Slattery, Wilson, Pawlowski)

- Collection of geometry-based data mapping algorithms for shared domain problems
- Data maps allow for efficient movement of data in parallel (e.g. between meshes of a different parallel decomposition)
- Ideally maps are generated at a desirable time complexity (logarithmic)
- Does not provide a general interface for all physics codes to couple to all other physics codes
- Does not provide discretization services (e.g. basis functions)
- Open-source BSD 3-clause license
- https://github.com/CNERG/DataTransferKit
Rendezvous Algorithm

- Initially developed by the Sandia SIERRA team in the mid-2000's for parallel mesh-based data transfer
- Creates parallel topology map that can be used repeatedly for data transfer
- Map execution uses asynchronous strategy (posts and waits) with minimal messages
- Effectively $N \log(N)$ time complexity for parallel topology map generation
- Relies on the generation of a secondary decomposition of the source and target meshes with a geometric-based partitioning (RCB)
DTK Implements Mappings for Various Transfers (Rendezvous used by all Mappings)

Shared Domain Map
Mesh → Point

Integral Assembly Map
Mesh → Geometry

Shared Volume Map
Geometry → Point

Aggregate cell contrib. to compute average in geometry

Colors represent different MPI processes
Example of a Data Transfer (Shared Domain Map)

Setup: Declare source geometric entities owned by each process:
- Entity global id
- Coordinate position/description

Declare target points required by each process:
- Cartesian points

Form rendezvous mesh and identify optimal data layouts for data movement (contiguous blocks for export/import ops)

Transfer: Given list of global ids and corresponding points, evaluate value at that point

Give me memory to fill that corresponds to points declared in setup
Data Transfer Kit
Weak Scaling Study (16 to 16K cores)

• Worst case scenario study (all-to-all) with 10K random points per core
  – Applications will have significantly better data locality

• Scaling study run on Titan

• Largest test problems so far over 1.0E9 elements and 1.0E5 cores

Excellent performance to 116K cores!
Transfer and Conservation
(Conservation is across pin sections)

Clad surface Temperature
\[ \text{Coefficients for heat flux} \]

SourceVolumeTransfer

\[ \text{BTU} / (\text{hr ft}^2 \text{ F}) \]
\[ \text{F} \]

\[ \text{Watts} / (\text{m}^2 \text{ K}) \]
\[ \text{K} \]

\[ \text{Watts} / (\text{m}^3) \]

\[ \text{K} \]

\[ \text{gm/cm}^3 \]

\[ \text{lb}_m/\text{ft}^3 \]

\[ \text{Fluid Temperature} \]
\[ \text{Fluid Density} \]
\[ \text{Power} \]

MOOSE

Insilico

CASL-U-2015-0108-000
Conservation

• DTK does not enforce conservation!
  – It identifies parallel mappings and transfers data

• How to implement conservation is usually discretization dependent

• For conservation DTK would need discretization information: mesh, topology, geometry, basis,…
  – CASL supports codes with completely different discretizations: CVFEM, CFEM, DGFEM, FD, FV, Geometric Control volume
  – Not feasible to support/duplicate discretization information for every kind of coupling.

• Conservation should be enforced by each application independently based on the code’s particular discretization
  – DTK can still transfer information/values to enforce conservation, but the code should do the enforcement since it knows the exact discretization scheme!
A Multiphysics Distributor
(Four levels of MPI communicators)
Significant Advances in Capability

- Successfully integrated MOOSE/Peregrine into VERA
- Successfully coupled MOOSE/Peregrine with Insilico
- Designed new multiphysics driver (Tiamat)
- Developed new data transfer mechanisms for specific applications
Virtual Reactor Integration

- We are successfully tackling significant software integration challenges
- This work requires a special combination of
  - Software design
  - PDE Solution techniques
  - Debugging skills (when Totalview, GDB, and Valgrind fail)
  - Build system and compiler knowledge