Code Coupling Libraries for High Performance Multi-Physics Simulation

SCD Seminar Series: Code Coupling

Philippa Rubin

4 May 2021

Acknowledgements

Software Outlook

- Support the UK's Collaborative Computational Projects (CCPs) and High-End Computing Consortia (HECs)
- Part of the Computational Science Centre for Research Communities (CoSeC), which is based within STFC.
- Have many projects, one of which has been to create a technical report on available code coupling libraries

https://www.softwareoutlook.ac.uk/

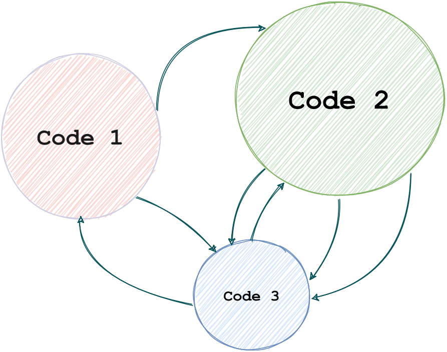
What is Code Coupling

Using multiple models

to solve a problem that

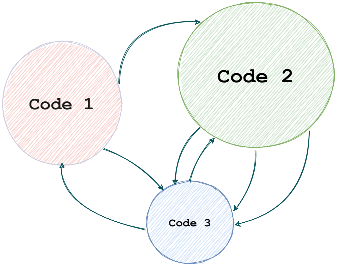
one model could not do

on its own

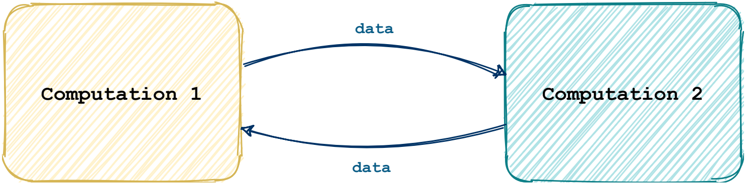


Why is Code Coupling of Interest

- Simulations that couple multiple physical phenomena is not a new idea
- Large-scale simulations require a framework to translate data between solvers and coordinate their separate calculations



Who Should be Interested in Code Coupling



Any developer who wants to do something like this!

Who Should be Interested in Code Coupling

Examples

- Shared domain problems, e.g. ocean / atmosphere model
- Piecing together two models that describe the same physical thing
- An application that does two things simultaneously with data dumps e.g. mechanics + statistical analysis

Traditional Picture

$$F_1(u_1, u_2) = 0$$

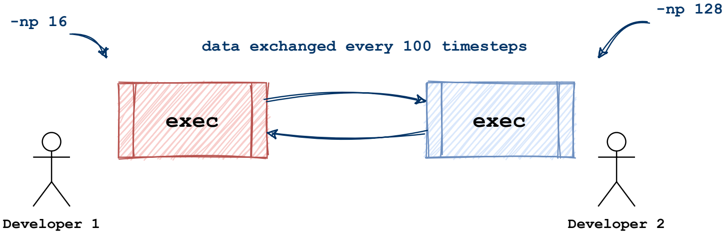
$$F_2(u_1, u_2) = 0$$

$$\partial_t u_1 = f_1(u_1, u_2)$$

$$\partial_t u_2 = f_2(u_1, u_2)$$

- Coupled evolution problem in (u_1, u_2), solved with a built-in JFKN solver or similar
- Many libraries appear limited to situations like this, as far as documentation suggests

General Applications



Could be as simple as two pieces of code running at the same time, with allocated resources. The above is quite easy to do with a coupler such as MUI.

Why you should use a coupling library

- Built to handle mass data send from one code to another
- Prevents building one monolithic executable where developers only know how pieces of it work
- Some of the libraries are very slick and easy to implement, can be $\sim$10 lines of code
- Easy to organise computing allocation to different parts of program

Traditional Example Demo

Problem Statement:

$$
\begin{aligned}
-\nabla \cdot \nabla u + \nabla v \cdot \nabla u &= 0 \\
-\nabla \cdot \nabla v &= 0
\end{aligned}
$$

Diffusion and Convection with u and v, with provided boundary conditions. We will solve this with **MOOSE**.

MOOSE

MOOSE

- Multiphysics Object-Oriented Simulation Environment
- Finite-element framework
- Developed by Idaho National Laboratory
- Very recently added training materials, virtual workshop on YouTube
- Lots of helpful material, anything from OOP to finite difference modelling

MOOSE Coupled Diffusion and Convection Demo

Write a short input file containing six things:

1. Mesh
2. Variables
3. Kernels
4. Boundary Conditions
5. Executioner
6. Outputs

MOOSE Coupled Diffusion and Convection Demo

Provide a Mesh

```
[Mesh]

    file = mug.e

[]
```

MOOSE Coupled Diffusion and Convection Demo

Coupling Variables

```
[Variables]
    [./convected]
        order = FIRST
        family = LAGRANGE
    [../]
    [./diffused]
        order = FIRST
        family = LAGRANGE
    [../]
[]
```

MOOSE Coupled Diffusion and Convection Demo

Kernels from Problem Statement

```
[Kernels]
    [./diff_convected]
        type = Diffusion
        variable = convected
    [../]
    [./conv]
        type = ExampleConvection
        variable = convected
        some_variable = diffused
    [../]
    [./diff_diffused]
        type = Diffusion
        variable = diffused
    [../]
[]
```

MOOSE Coupled Diffusion and Convection Demo

Boundary Conditions

```
[BCs]
    [./bottom_convected]
        type = DirichletBC
        variable = convected
        boundary = ‘bottom’
        value = 1
    [../]
    [./top_convected]
        type = DirichletBC
        variable = convected
        boundary = ‘top’
        value = 0
    [../]
```

MOOSE Coupled Diffusion and Convection Demo

Boundary Conditions continued

```
    [./bottom_diffused]
        type = DirichletBC
        variable = diffused
        boundary = ‘bottom’
        value = 2
    [../]
    [./top_diffused]
        type = DirichletBC
        variable = diffused
        boundary = ‘top’
        value = 0
    [../]
[]
```

MOOSE Coupled Diffusion and Convection Demo

Ask for a solver

```
[Executioner]
    type = Steady
    solve_type = ‘PJFNK’
[]
```

MOOSE Coupled Diffusion and Convection Demo

How you want to output

```
[Outputs]
    execute_on = ‘timestep_end’
    exodus = true
[]
```

MOOSE Coupled Diffusion and Convection Demo

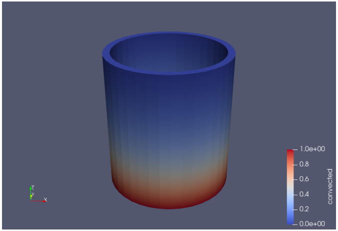


Figure 1: Convected Variable

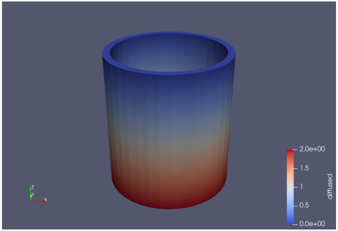


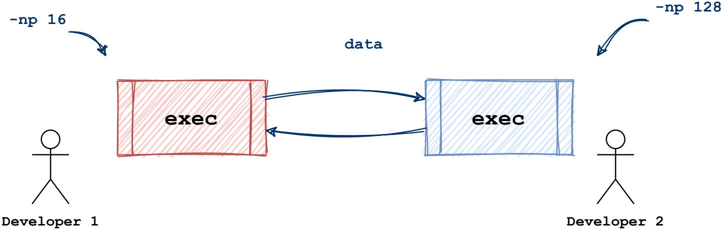
Figure 2: Diffused Variable

MOOSE Summary

MOOSE

- Plenty of training materials, easy to learn
- Consistent; provided conda environment works well
- Restricted to ‘traditional’ picture of code coupling
- Difficult to see how to add MOOSE to existing codebases

General Example

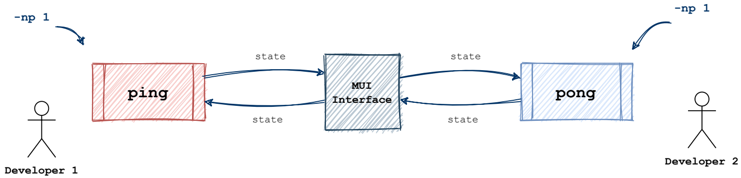


Coupling doesn't have to be scientific, can be as simple as two executables that want to share data between each other

MUI

- Multiscale Universal Interface
- Originally developed by Brown University, maintained today by STFC
- Helpful demos available on GitHub, recent workshop as part of an IROR training series
- Easiest to learn to use in the Software Outlook project

Ping Pong MUI Example



Two executables: `ping` and `pong`. Fire off values back and fourth to the MUI Interface

Ping Pong MUI Example

Listing 1: ping.cpp

```
#include "mui.h"

int main() {

  mui::uniface1d interface( "mpi://ping/ifs" );
  mui::sampler_exact1d<int> spatial_sampler;
  mui::chrono_sampler_exact1d chrono_sampler;
  mui::point1d push_point;
  mui::point1d fetch_point;
```

Clone MUI, include header, configure interface (template configs available)

Ping Pong MUI Example

Listing 2: pong.cpp

```cpp
#include "mui.h"

int main() {

  mui::uniface1d interface( "mpi://pong/ifs" );
  mui::sampler_exact1d<int> spatial_sampler;
  mui::chrono_sampler_exact1d chrono_sampler;
  mui::point1d push_point;
  mui::point1d fetch_point;
```

Clone MUI, include header, configure interface (template configs available)

Ping Pong MUI Example

Listing 3: ping.cpp

```
  int state = 0;
  for ( int t = 0; t < 10; ++t ) {
    state++;
    push_point[0] = 0;
    interface.push( "data", push_point, state );
    printf( "Ping sending: %d\n", state);
    interface.commit( t );
    fetch_point[0] = 0;
    state = interface.fetch( "data", fetch_point, t, spatial_sampler,
    chrono_sampler );
    printf( "Ping receives: %d\n", state);
  }
  return 0;
}
```

Ping Pong MUI Example

Listing 4: pong.cpp

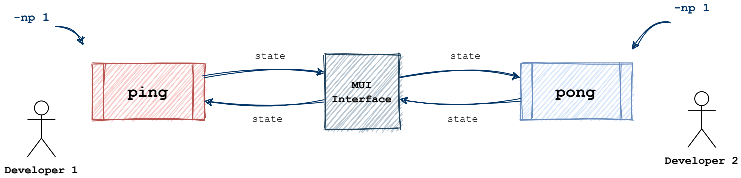
```
  int state;
  for ( int t = 0; t < 10; ++t ) {
    fetch_point[0] = 0;
    state = interface.fetch( "data", fetch_point, t, spatial_sampler,
    chrono_sampler );
    printf( "Pong receives: %d\n", state);
    state--;
    push_point[0] = 0;
    interface.push( "data", push_point, state );
    interface.commit( t );
    printf( "Pong sends: %d\n", state);
  }
  return 0;
}
```

Ping Pong MUI Example

Listing 5: ping pong with MUI

```
mpic++ -std=c++11 -O3 ping.cpp -o ping
mpic++ -std=c++11 -O3 pong.cpp -o pong
mpirun -np 1 ./ping : -np 1 ./pong
rank 0  identifier mpi://ping/ifs   domain size 1   peer number 1
rank 1  identifier mpi://pong/ifs   domain size 1   peer number 1
Ping sending: 1
Pong receives: 1
Pong sends: 0
Ping receives: 0
Ping sending: 1
Pong receives: 1
Pong sends: 0
Ping receives: 0
...
```

Ping Pong MUI Example



`ping` and `pong` could have been anything in this MUI example. Can send and receive large amounts of data to and from the MUI Interface easily

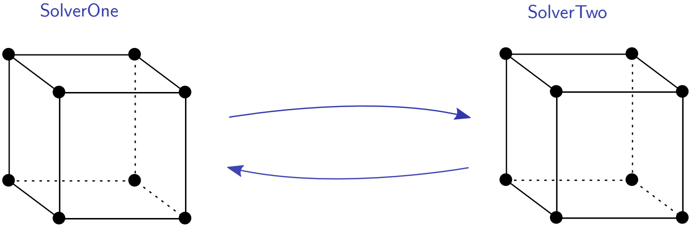
MUI Summary

- Easiest to learn, quick to implement
- Helpful demos available on GitHub, recent workshop as part of an IROR training series
- No dependencies, consistent
- Easy to design MUI implementation for pre-existing codebases

preCICE

- Precise Code Interaction Coupling Environment
- Particular interests in fluid-structure interaction and conjugate heat transfer simulations
- Developed by doctoral candidates from the Technical University of Munich and the University of Stuttgart.
- Has training materials, have to be selective about this, more verbose coupler

preCICE solver dummies



Very similar to the MUI `ping` `pong` example, but have to specify a mesh in configuration

preCICE Configuration: Mesh and Data

```xml
<solver-interface dimensions="3">
  <data:vector name="dataOne" />
  <data:vector name="dataTwo" />
  <mesh name="MeshOne">
    <use-data name="dataOne" />
    <use-data name="dataTwo" />
  </mesh>
  <mesh name="MeshTwo">
    <use-data name="dataOne" />
    <use-data name="dataTwo" />
  </mesh>
```

preCICE Configuration: Participants

```
<participant name="SolverOne">
  <use-mesh name="MeshOne" provide="yes" />
  <write-data name="dataOne" mesh="MeshOne" />
  <read-data name="dataTwo" mesh="MeshOne" />
</participant>
```

preCICE Configuration: Participants

```
<participant name="SolverTwo">
  <use-mesh name="MeshOne" from="SolverOne" />
  <use-mesh name="MeshTwo" provide="yes" />
```

preCICE Configuration: Participants

```
  <mapping:nearest-neighbor
  direction="write"
  from="MeshTwo"
  to="MeshOne"
  constraint="conservative" />
  <mapping:nearest-neighbor
  direction="read"
  from="MeshOne"
  to="MeshTwo"
  constraint="consistent" />
  <write-data name="dataTwo" mesh="MeshTwo" />
  <read-data name="dataOne" mesh="MeshTwo" />
</participant>
```

preCICE Configuration: Communication and Coupling Scheme

```
    <m2n:sockets from="SolverOne" to="SolverTwo" />
    <coupling-scheme:serial-implicit>
      <participants first="SolverOne" second="SolverTwo" />
      <max-time-windows value="2" />
      <time-window-size value="1.0" />
      <max-iterations value="2" />
      <min-iteration-convergence-measure min-iterations="5"
      data="dataOne" mesh="MeshOne" />
      <exchange data="dataOne" mesh="MeshOne"
      from="SolverOne" to="SolverTwo" />
      <exchange data="dataTwo" mesh="MeshOne"
      from="SolverTwo" to="SolverOne" />
    </coupling-scheme:serial-implicit>
  </solver-interface>
</precice-configuration>
```

preCICE solver dummies

Listing 6: solverdummy.cpp

```
for (int i = 0; i < numberOfVertices * dimensions; i++)
{
  writeData.at(i) = readData.at(i) + 1;
}
```

preCICE solver dummies

Open two terminal windows, run

```
./solverdummy ../precice-config.xml SolverOne MeshOne
```

```
./solverdummy ../precice-config.xml SolverTwo MeshTwo
```

preCICE Summary

- Very capable, can configure to do many things
- Much more verbose to learn and implement
- Main training materials are not very helpful to new users

Other Couplers Considered: OpenPALM

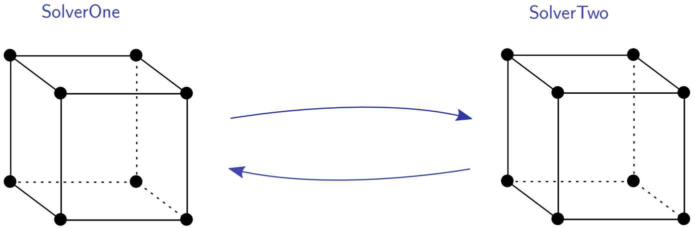
- Projet d'Assimilation par Logiciel Multimethodes
- Joint team between Cerfacs and Onera, Cerfacs also created OASIS
- Comes in two parts, prePALM and PALM
- Very unreliable in this project. Unexpected behaviour with communication between domains

Other Couplers Considered: PLE

CODE SATURNE

- Point Location Exchange library
- Part of CodeSaturne, CFD software released by EDF
- Another communication framework for code coupling similar to MUI
- Difficult to recommend to a new user, lack of documentation in comparison to others

Comparing Performance



As coupling essentially comes down to data exchange, can compare performance with a field exchange example implemented in each coupler one by one

Software Outlook Technical Report

- Links to most useful training materials, demos, usability recommendations
- Performance comparison with 3D Field Exchange example

Comparison of Code Coupling Libraries for High Performance Multi-Physics Simulation

Philippa Rubin*

Science and Technology Facilities Council, Hartree Centre, Daresbury Laboratory, Daresbury, Warrington, WA4 4AD, United Kingdom

March 2021

Abstract

The usability and performance of code coupling libraries are compared. In this work, MUI, MOOSE, preCICE, OpenPALM and PLE are considered. To compare performance, a 3D field exchange example was provided by the Scientific Computing Department at STFC. It is intended that this is a sufficiently general example to ensure this work applied to a wider range of the UK's Computational Collaborative Projects and High-End Computing Consortia.

1 Introduction

As available computational power increases, there is a growing interest in the use of large-scale multi-physics solvers in a range of life science, physical science and engineering projects. In most cases, a single solver will not offer all of the required capabilities and so a coupling framework is required to translate data between solvers and co-ordinate their separate calculations. A number of code coupling libraries have been developed, and over the course of this project's work many of them ran their first ever training workshops and have gained more and more support. Many have reached a level of maturity for us to compare their attributes and performance when used on large-scale problems. This document walks through some of the available code coupling libraries, followed by a performance comparison with a 3D field exchange example.

*philippa.rubin@stfc.ac.uk

1

Code Coupling Libraries for High Performance Multi-Physics Simulation

SCD Seminar Series: Code Coupling

Philippa Rubin

4 May 2021