University of Tennessee, Knoxville
Trace: Tennessee Research and Creative Exchange

Masters Theses | Graduate School

8-2013

An Interoperable Executive Library for Loosely Coupled Physics Systems

Andrew Austin Kail
University of Tennessee - Knoxville, akail@utk.edu

This Thesis is brought to you for free and open access by the Graduate School at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Andrew Austin Kail entitled "An Interoperable Executive Library for Loosely Coupled Physics Systems." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Aerospace Engineering.

Kwai L. Wong, Major Professor

We have read this thesis and recommend its acceptance:

A. J. Baker, Xiaopeng Zhao

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

An Interoperable Executive Library for Loosely Coupled Physics Systems

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Andrew Austin Kail

August 2013

Copyright © 2013 by Andrew Kail

All rights reserved

Acknowledgments

I would like to express deep gratitude to Dr. Kwai Wong for his support and guidance in my work in graduate school. I would also like to thank Dr. A.J. Baker and Dr. Xiaopeng Zhao for serving on my Committee and for their assistance in the completion of this thesis.

I would also like to recognize the previous work done by Keith Seymour, Xia Henian, Elton Freeman, and Dr. Kwai Wong.

I would like to express my appreciation to the Joint Institute for Computational Sciences and the support that I have received from them. Special thanks to Christian Halloy and Angie Chance for all their help at JICS. And thank you to my colleagues Elton Freeman and Mikhail Sekachev for their support and continued friendship.

Abstract

The complexity of simulating any system-wide process involving biomedical processes or thermal-fluid systems goes beyond the reach of a single computer code. In this document I present an Interoperable Executive Library (IEL) that has been designed to run, in parallel, a collection of multi-component physics simulations. The IEL is a light-weight integrator responsible for managing the distribution of data and memory, coordinating communication among parallel processes, and direct execution of a set of loosely coupled numerical and physics tasks HPC resources

Presented are two case studies utilizing the IEL. The first case simulates conjugate heat transfer coupled with a potential flow solver for convective properties and a radiosity module for radiation exchange. Radiation analysis is offloaded to GPGPU's and includes source terms for time variant solar flux. This simulation illustrates the ability of the IEL to schedule multiple solvers and the subsequent data transfer required.

The second test case involves the electro-mechanical simulation of the heart with an associated fluid flow handled by an Incompressible Navier-Stokes solver. In this example I demonstrate the capability of the library to efficiently integrate third-party software and handle more advanced scheduling techniques. The results of the simulations obtained from running on Kraken (CRAY XT5 at NICS), Keeneland (HP SL250G8 at Georgia Tech and NICS) and STAR (Dell T7500 at UT) will be examined and shown.

Table of Contents

1 Introduction **1**

1.1 The IEL 2

1.2 Physics Modules 3

1.3 My Contributions 4

2 The Interoperable Executive Library **6**

2.1 The Executive 8

2.2 Driver Program 9

2.3 Configuration File 11

2.4 Communicator Library 13

2.5 External Tools 15

2.5.1 Pre-Processing 15

2.5.2 Post-Processing 17

2.5.3 Solvers 17

3 Loosely Coupled Physics Systems **20**

3.1 Thermal-Fluid Test Case 21

3.2 Biomechanical Test Case 24

3.3 Computational Implementation 28

4 Fluid Modules **33**

4.1 Governing Equations 33
4.2 Finite Element Implementation 36
4.3 Module Structure 44

5 Thermal Module **46**
5.1 Governing Equations 46
5.2 Finite Element Implementation 48
5.3 Module Structure 50
5.4 Test Case Results and Discussion 50

6 BioMechanical Module **54**
6.1 Governing Equations 54
6.2 Finite Element Implementation 57
6.2.1 Mechanical Model 57
6.2.2 Electrophysiology Model 59
6.2.3 Computational Implementation 60
6.3 Module Structure 61
6.4 Mesh Module 63
6.5 Reaction-Diffusion Module 63
6.6 Test Case Results and Discussion 66

7 Conclusion **69**

Bibliography **71**

Appendix **75**

Appendix A IEL Definitions **76**
A.1 Data Structures 76
A.1.1 Detailed Description 77

A.2 Communicator Library 78

Appendix B Test Case Drivers and Configuration Files **81**

B.1 Thermal-Fluid Test Case 81

B.2 Biomechanical Test Case 83

Appendix C Thermal Module Formulation **85**

Appendix D ODE Models **87**

D.1 Beeler-Reuter Model 87

Vita **97**

List of Tables

D.1 Table: Rate constants for α and β 89

List of Figures

2.1 An overview of the IEL showing the interaction between the driver, configuration file, executive and communicator library 7
2.2 Example of the executive during runtime with multiple processes and interprocess communication 9
2.3 Basic driver example loading and executing two modules with different configuration files 10
2.4 Single module driver example written in the C language 11
2.5 Configuration file demonstrating the simultaneous used of two modules . . . 12
2.6 Illustration of the communicator libraries role in sharing data between multiple physics modules for a thermal-fluid system 13
2.7 Communicator library example showing the methods used to distribute and send data 14
2.8 Heart model partitioned into eight separate domains using the PT 15
2.9 Example models and meshes as generated with Cubit 16
2.10 Paraview visualizing the heart model and the resultant stimulation and distortion 18

3.1 Multi physics system for conjugate heat transfer, radiation and fluid flow . . 21
3.2 Results obtained for the thermal-fluid test case on the cubi module 22

3.3 Computational workflow thermal-fluid system workflow using the fluid, radiosity and thermal modules in a multi-physics setting 24
3.4 Simulation of the biomechanical system on the ventricles of a heart model . . 26
3.5 Workflow for the biomechanical test case using the biomechanical, mesh, preprocessing and fluid modules 27

4.1 Overview of the structure and flow of the fluid modules 45

5.1 Thermal module structure and flow including shared boundary modules . . . 51
5.2 Results from the potential flow solver on the cubi model 52
5.3 Final results from the thermal module solution of the cubi model at t=0.0 sec 52
5.4 Final results from the thermal module solution of the cubi model at t=30.0 sec 53
5.5 Final results from the thermal module solution of the cubi model at t=60.0 sec 53

6.1 Results from Beeler-Reuter model run in a standalone MATLAB program . . 58
6.2 Structure and flow of the biomechanical module 62
6.3 Simulation of a basic heart model using the Reaction-Diffusion Module . . . 65
6.4 Original geometries used for the biomechanical test case 66
6.5 Simulation at t = 5 ms 67
6.6 Simulation of cardiac tissue using realistic geometry 68

D.1 Results from Matlab simulation of Beeler-Reuter Model 91

Nomenclature

α, β	gate open/close rate
β	volumetric thermal expansion coefficient
ϵ	thermal emissivity
η	convection switching term
μ	kinematic viscosity
ϕ	scalar potential function
Ψ	trial space Function
ρ	density
$\{N\}$	shape function
C	continuity constraint
G	Galerkin weak statement
Ca^{2+}	intracellular calcium concentration
C_m	transmembrane capacitance
C_p	heat capacity at constant pressure
h_{conv}	convection
b	body force
D	conduction tensor
d	calcium activation gate
E,F	Galerkin tangent terms
f	calcium inactivation gate

Gr	Grashoff number
h	sodium inactivation gate
I	ion channel
j	sodium inactivation gate
M	4th order spatial tangent moduli
m	sodium activation gate
Nu	Nusselt number
P	pessure
Pr	Prandtl number
Q	sensitivity to Kirchoff stress
Re	Reynold number
T	temperature
u,v,w	velocities in x,y and z directions
V	transmebrane action potential
x	potassium activation gate

Chapter 1

Introduction

Solutions of multiphysics systems have taken many leaps forward in the past several decades, in particular in the last few years. With the advancement of computational power in systems such as Kraken (Cray XT5 at NICS), Sequoia (Blue/Gene Q at NCSA) and the K computer (SPARC64 VIIIfx in Kobe, Japan), and hybrid computing machines like Keeneland (HP SL250G8 at Georgia Tech and NICS), Titan (Cray XK7 at ORNL), and Tianhe-2(China), frameworks are being developed to handle large computationally expensive tasks.

A multitude of solvers and software frameworks have been developed over the years, each using different techniques and methods with multiple purposes in mind. ROCCOM is one of the first frameworks developed to handle data transfer in multiphysics simulations for rocket motor designs. ROCCOM organizes its data into objects called windows which can then be split into panes for parallel computation. The physics modules then reference the windows and panes to handle data transfer[1].

Another solver is COMSOL, a finite element multiphysics simulation environment. COMSOL brings the pre-, post-processing and solvers under the control of one system with a graphical user interface [2]. DUNE is also a problem solving environment that can solve PDE systems using finite volume, finite elements and finite difference methods[3].

OpenFOAM is a project built as an open source framework for computational fluid dy-

namics. OpenFOAM has matured over the years into a suite of solvers covering combustion, structural mechanics, heat transfer and electromagnetics[4]. Another framework used in the scientific community is CACTUS, which is a parallel toolkit for use in studies on numerical relativity. CACTUS also employs the use of plugins called "thorns" that expands its scope of application[5].

LibMesh is a problem solving environment and framework for the solution of partial differential equations. Like COMSOL libMesh uses finite element formulations for the solution of PDE systems [6]. Similar to libMesh is FEniCS, a collection of interoperable applications that are accessed through a user interface named DOLPHIN [7]. Both of these problem solving frameworks make use of modern parallel solvers such as those available in PETSc and Trilinos.

A fluid structure interaction study involving multiphysics simulations was also performed by the Karlsruhe Institute of Technology in Germany using the Fluent fluid solver from ANSYS and the finite element structural solver ABAQUS. This simulation was developed to replicate, in 3D, cardiovascular blood flow through the heart and utilizes an external communication program to transfer boundary information [8].

1.1 The IEL

While these software packages have continued to push the envelope of physics simulations the Interoperable Executive Library, henceforth IEL, is designed with a different approach in mind. The goal is to efficiently incorporate physics solvers in a modular fashion and provide a simple application programming interface, or API, for handling data transfer and scheduling. It includes a number of sparse and dense parallel solvers from the Trilinos[9], MAGMA[10], ScaLAPACK[11], and PETSc[12] libraries. In addition, the IEL has been built with a set of pre- and post-processing units based on the I/O format of HDF5 and PATRAN in conjunction with Cubit[13] and Paraview[14].

Of the several methods used for the solution of multiphysics systems, two common approaches are the monolithic and the loosely coupled method. Monolithic approaches to physics simulations involve using one solver to handle the solution of the governing equation sets simultaneously. The monolithic method is very straightforward and relatively simple, but does have drawbacks. One major drawback is handling the solution of hybrid sparse and dense matrix systems concurrently. For instance, the conjugate heat transfer problem demonstrated later requires the solution of a sparse matrix for the governing equations for conduction while a dense matrix solution is also required for the radiation exchange. Solving these systems under one system ($Ax = b$) reduces the efficiency of the solvers used.

The loosely coupled method alleviates some of the issues by dividing the work load between multiple solvers. A feature of this method involves the use of shared boundary conditions, which act as points of data exchange between the two solvers. The IEL utilizes this loosely coupled method, but due to its modular nature it can also run monolithicly designed programs. This, coupled with configuration files allows the IEL to run multiple simulations either simultaneously or in sequence and they can then be scaled and implemented in various ways for simultaneous execution utilizing various configurations for parametric studies.

The biggest advantage of the IEL is the ability to schedule multiple physics tasks in parallel. This allows the simulation to be conducted on medium and large scale computing platforms such as Keeneland with GPU's and Kraken with over 100,000 cores. This also requires portability across machines. The IEL is designed to compile and run on every type of computing system and can even run on machines using hybrid GPU accelerators.

1.2 Physics Modules

The modules utilized in this thesis have all been developed and verified independently and make use of finite element numerical schemes. The primary physics modules are also supported by an array of pre and post processing modules for mesh generation and domain

decomposition.

The thermal module consists of two components for handling the solution of conjugate heat transfer. The first component provides the solution for a radiative heat transfer between radiating surfaces. The second component utilizes information provided by a fluid flow solver to determine convective properties of the exposed surfaces and then solves the system using standard energy conservation principles.

The biomechanical module has been developed to simulate the electrophysiological responses of cardiac tissue. The model works by replicating the stimulation, activation of ionic current gates and the subsequent reaction-diffusion of the tissue. The heart model has been divided into three distinct parts. The ordinary differential equation (ODE) for the action-potential of cardiac tissue, the reaction-diffusion equation and a stress model.

Both the preceding modules require interaction with a fluid module. The fluid module has been designed in two parts. The first portion is built for simulating simple potential flow and is used for generating surface velocity distributions in the thermal module. A solution of the incompressible Navier-Stokes equations is handled by the second solver and is used for simulating the internal flow characteristics of the heart model as generated by the biomechanical module.

1.3 My Contributions

The IEL was originally developed at the University of Tennessee by Keith Seymour and Dr. Kwai Wong. In this document I present the following contributions to the development and expansion of the Interoperable Executive Library and its components.

(a) Expand the functionality of the IEL to incorporate new solvers.

(b) Build modules for Potential and Incompressible Navier-Stokes flow using the finite element method.

(c) Develop a mesh module to generate a 3D mesh from a 2D mesh to facilitate the interaction of physics modules.

(d) Design new Electrophysiological module for the heart simulation.

(e) Showcase the IEL by integrating multiple physics modules into example case studies.

(f) Present the results of the simulations as performed on a set of parallel computers.

Chapter 2

The Interoperable Executive Library

The IEL is a software framework used for multiphysics simulations and is designed to execute and schedule in parallel a series of physics solvers. In these multi-physics simulations domain interaction is a common occurrence and therefore requires data and information exchange on points called shared boundaries. The IEL is designed to initiate and manage the data exchange for these types of simulations.

Beyond its scheduling and data managing capabilities the IEL also makes use of common scientific libraries such as Trilinos, MAGMA and Hips. Other tools for grid generation using Cubit and visualization with Paraview have also been utilized. By including these tools into modules already using the IEL provides a guide for building new software with them. Integration of third party solvers as modules is an important functionality of the IEL. By integrating new software it is possible to extend the scope of the application of the library.

In figure 2.1 is an overview of a simulation using the IEL during runtime. A driver program initiates the execution of the simulation by passing a configuration file to the executive. There the executive reads the configuration file and begins the simulation with two separate modules with communication in between.

The IEL is built as a library it must therefore be used from within an external program. In this document this program is called a "driver" which will load the module and execute the

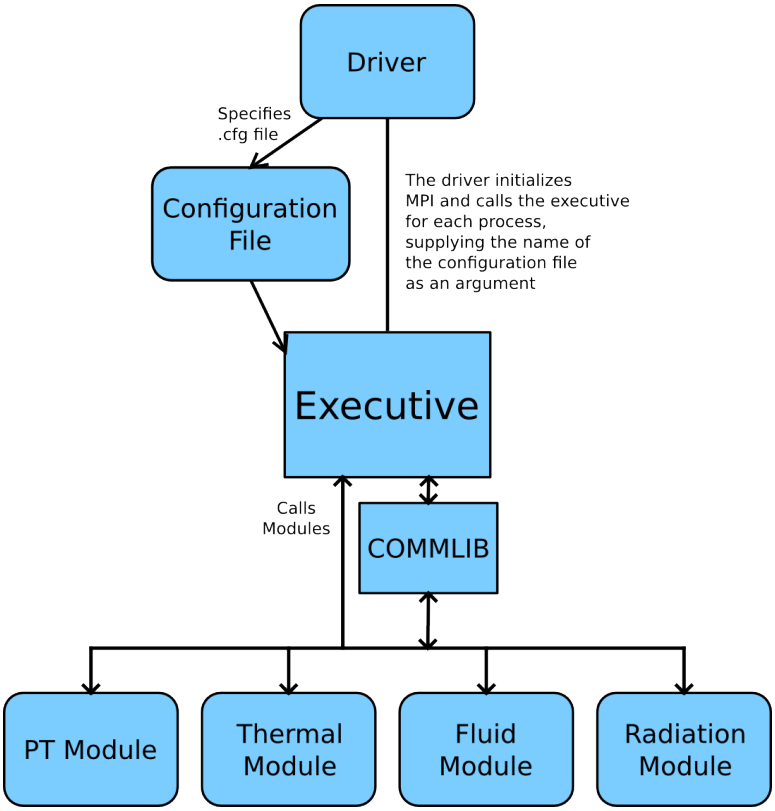


Figure 2.1: An overview of the IEL showing the interaction between the driver, configuration file, executive and communicator library

IEL. The library itself consists of three components: the configuration file, the communicator library and the executive. The configuration file defines the functionality of each simulation, the number of shared boundary conditions between different modules, and the number of processors that will be assigned for the parallel simulation.

The second component is the communicator library (COMMLIB) acting as the interface for data exchange. The communicator is built as a wrapper for the Message Passing Interface (MPI) and handles the transfer of the data. It is through this communication library that the information for the shared boundary conditions is exchanged between modules.

The third component is the executive. The executive, in cooperation with COMMLIB,

acts as a scheduler and data manager and its behavior is guided by the information provided in the configuration file. The configuration file tells the executive how much memory to allocate for the shared boundary conditions, which modules to execute, and the number of processes assigned to each module.

The applications of the IEL extends into many scientific fields, most specifically in the field of computational sciences. The library allows for large scale physics simulations to be conducted on modern computing hardware. It also has the ability to extend into new and emerging computing architectures such as Intel®Xeon phi™coprocessor and GPGPU's.

2.1 The Executive

The executive is the underlying framework for the IEL. It manages the distribution of the memory, coordinates the communication via COMMLIB and through the driver executes the simulation. The executive is used to load and initialize the simulation and is an interface point within a main program for executing the simulation.

The executive works in parallel using a standard and portable library implementation, which is demonstrated in figure 2.2. This figure shows a series of process running in parallel and communicating among themselves through the executive and the communicator library. Through the functions provided the executive is able to manage each of the process, sending data in a loop.

At the execution of the simulation the executive reads a configuration file providing information on memory allocations, communicators, input arguments and shared boundary data distribution. It has a direct interface within the physics modules to define a set of data structures which holds information of the shared boundary data, communicators, and input arguments.

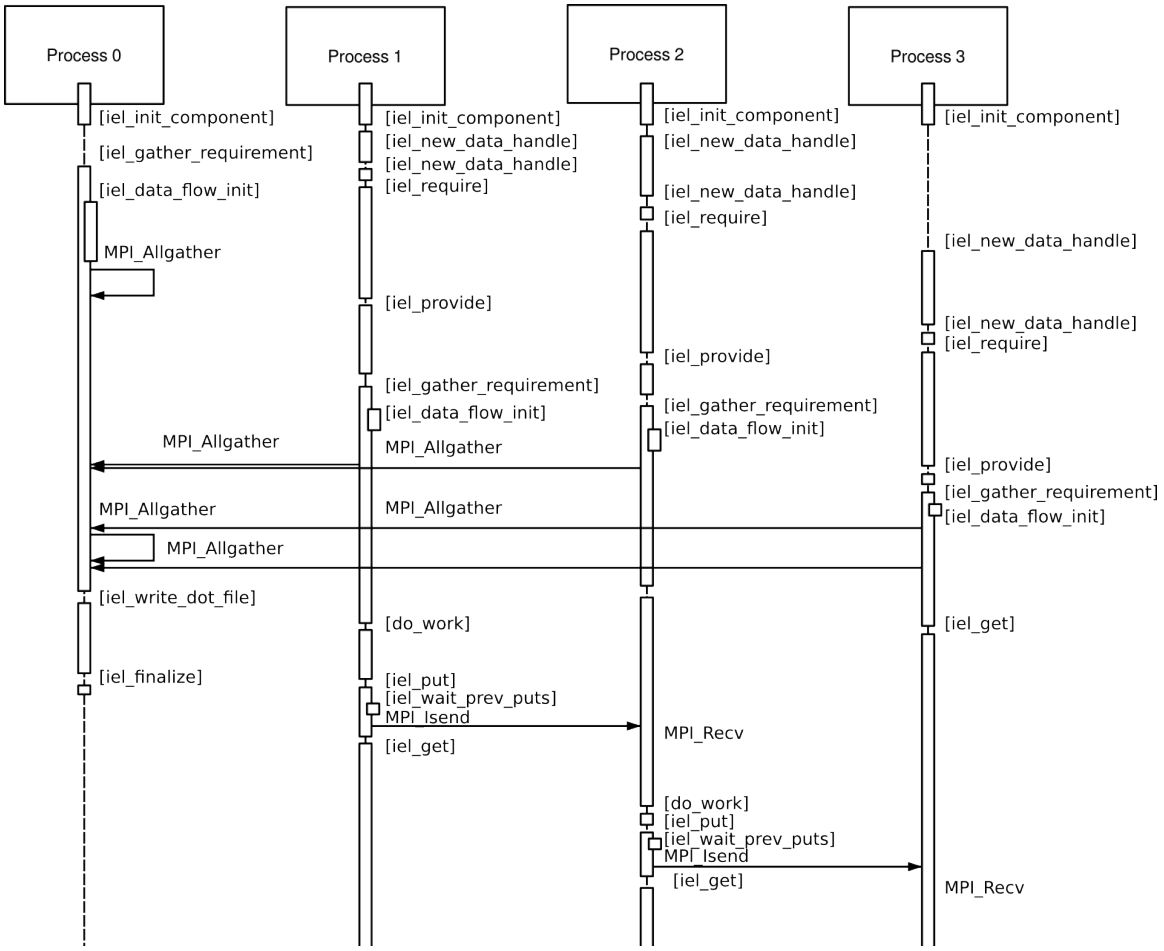


Figure 2.2: Example of the executive during runtime with multiple processes and inter-process communication

2.2 Driver Program

The driver is responsible for the scheduling of the simulation and provides the means for managing the solution of complex multi-physics systems. It is the driver that loads the modules and executes the simulation.

An example of the workflow and execution of a simple driver program is illustrated in figure 2.3. In this figure a parallel simulation is initialized, then two modules are executed in

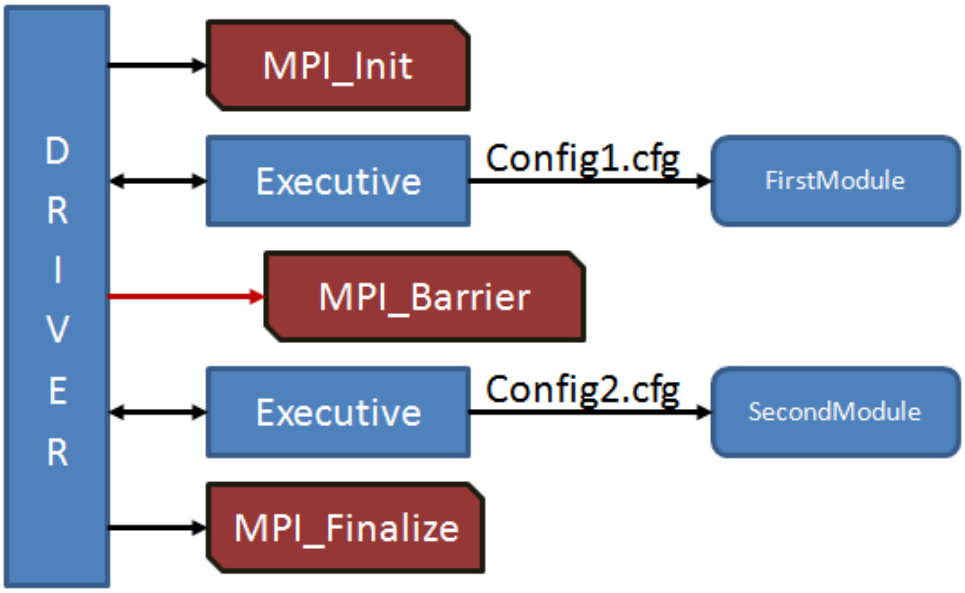


Figure 2.3: Basic driver example loading and executing two modules with different configuration files

succession using individual configuration files. The driver however is not limited to simply adding modules and executing a simulation in a serial fashion. Because the scheduling is handled in the driver it can be built with multiple combinations of modules and configuration files. Execution of a simulation does not have to occur in a parallel fashion, but can instead be integrated inside control structures, such as loops and

or if statements, providing control over the parameters and input data.

The driver works by loading modules as external libraries and initiating execution by the functions provided by the executive. Simple driver programs load the modules and execute the simulations in a serial fashion. However, the driver gives room for more complex simulations to be performed as will be shown with the second case study.

A one module example driver written is C is shown in Fig 2.4. At this point the driver loads a module using the add module function and starts the simulation by passing the

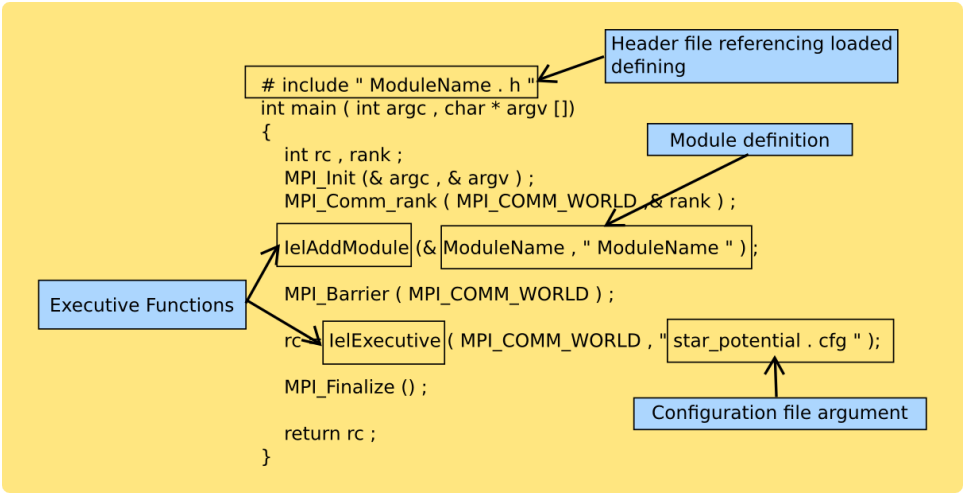


Figure 2.4: Single module driver example written in the C language

configuration file to the executive. Once the executive completes the simulation a value indicating the status of success or failure is returned.

2.3 Configuration File

The configuration file is a user defined file that is passed into the executive at runtime and provides all of the necessary information for the simulation. One of the primary data fields provides information on the shared boundary points specifying the size of boundaries shared between modules, which will be discussed in detail in Chapter 3. Also in the configuration file the input arguments, the number of processes, the library type, and the per module memory size of the shared boundary data of each module are defined. This allows the executive to pre-allocate memory and distribute data accordingly.

The configuration file provides the means to define the simulation in an easy to read text based format. Multiple configuration files can be can be used within the driver. Because the configuration file and executive support multiple module execution, the same module can

```
shared_bc_sizes = [149999,149999,3,3]

modules = (
        {
                function="data_module";
                library="static";
                args();
                size=2;
                points=(
                (([0,149999,(),([0,3]),()),
                ((),([0,149999,(),([0,3]))
                );
        };
        {
                function="main_function";
                library="static";
                args("./libexec.so","-da_grid_x","100000","-log_summary",
                "-snes_monitor","ksp_monitor");
                size=2;
                points=(
                (([0,149999,(),([0,3]),()),
                ((),([0,149999,(),([0,3]))
                );
        };
);
```

A list of the maximum size of each BC in the config file

The name of the function for this module

The type of library for this module

A list of the arguments that are passed to this module at initialization. Similar to argv command line arguments

The number for processors this module uses

Point allocations based on the processor and size of the sbc

Figure 2.5: Configuration file demonstrating the simultaneous used of two modules

be called several times during one execution with or without shared boundary points. This simultaneous execution can be beneficial for parametric studies.

An example configuration file is presented in figure 2.5 which illustrates a file defining multiple modules with shared boundary points. On the first line the maximum size of the shared boundary arrays is defined. Then four arrays are defined and referenced again in the points section of each module. Each module is defined by its name, library type and input data. Input arguments for the module are specified in the configuration file and passed to the executive which runs the simulation.

2.4 Communicator Library

The communicator library is the interface for data exchange and serves to initialize and declare the data handles used as the points for data exchange. After the data is initialized the modules are then free to communicate during the simulation at will. An illustration of how COMMLIB interacts with the other components of the IEL can be found in figure 2.6. In this example the three modules of the thermal-fluid system execute on their own set of processes. Among the thermal module, the fluid module, and the radiosity module COMMLIB handles the exchange of data on the shared boundaries.

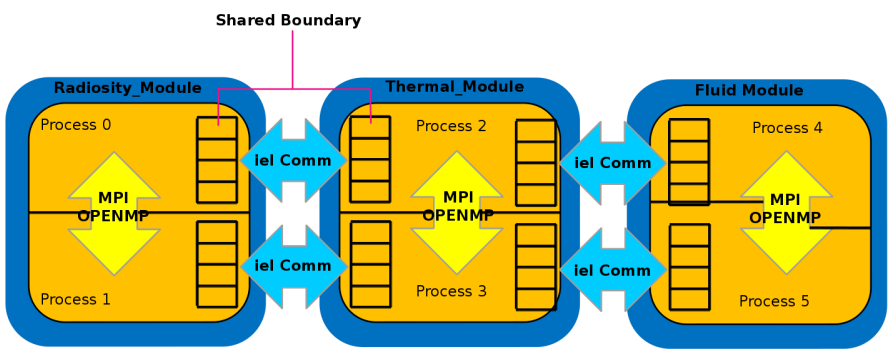
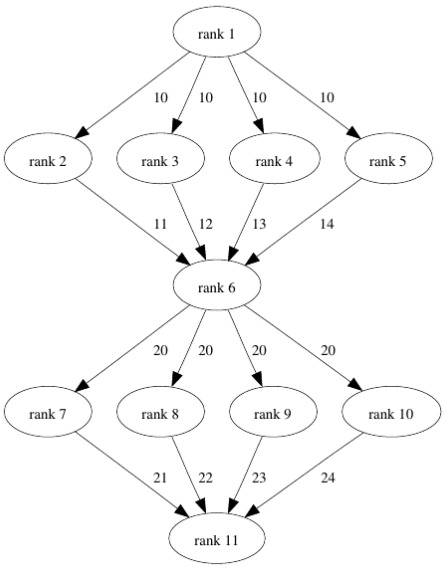


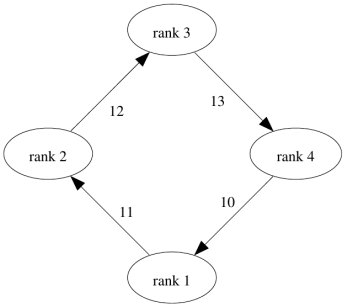
Figure 2.6: Illustration of the communicator libraries role in sharing data between multiple physics modules for a thermal-fluid system

COMMLIB operates with a series of functions that are built as wrappers for MPI protocols. The two most common functions are the *iel_get* and *iel_put* functions. These functions send and request the data handles used as input. *iel_put* is typically a non-blocking function but may become blocking if a previous call to put has not completed. *iel_get* on the other hand is always a blocking function. A full list of the communicator library functions can be

found in Appendix A.



(a) Example of sending data out to multiple processes then converging back to one



(b) Example sending data through multiple processes in a ring

Figure 2.7: Communicator library example showing the methods used to distribute and send data

In figure 2.7 are two test cases using COMMLIB as a communication tool. Figure 2.7a shows the master process distributing the data out to all other processes using *iel_put*. The data is then returned to the master process using a series of *iel_get* and repeated as needed. Figure 2.7b is a simple ring test, using a series of put and get functions to send the data in a ring around the processes.

2.5 External Tools

2.5.1 Pre-Processing

The PT

The PT is a preprocessing tool developed to read in the Patran or HDF5 file formats, parse the data, then return the input files for use in certain physics solvers. These files are designed to provide geometrical and mathematical information in a format suited for parallel computation.

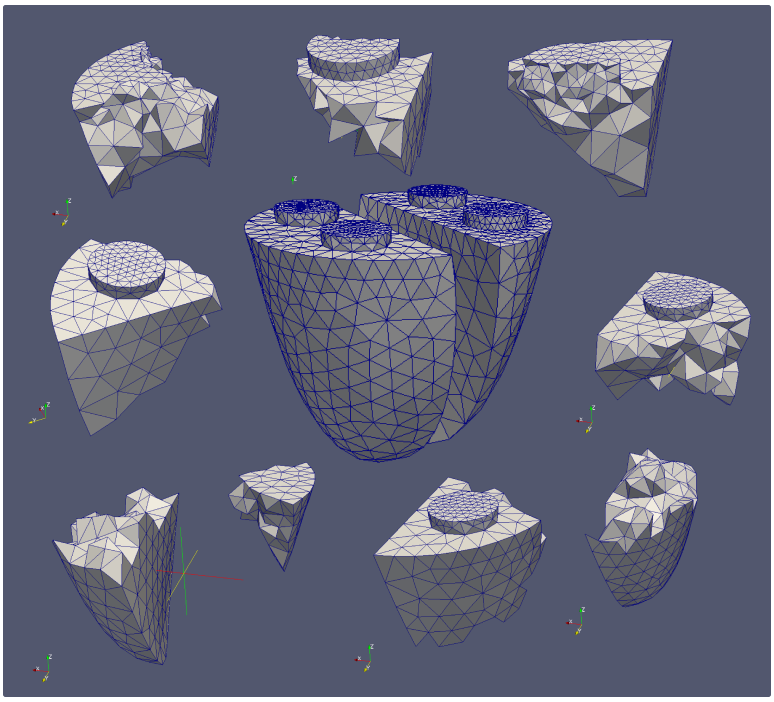


Figure 2.8: Heart model partitioned into eight separate domains using the PT

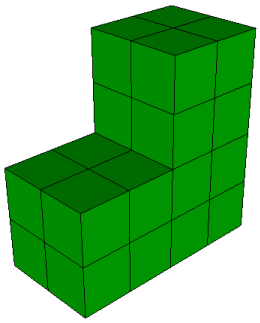
Domain decomposition is performed using the METIS library. As shown in figure 2.8 the PT uses METIS to partition the input mesh into multiple sections. During the simulation each process generates the solution for its part of the domain. At the end of the simulation

the parts can be re-assembled back to the original image.

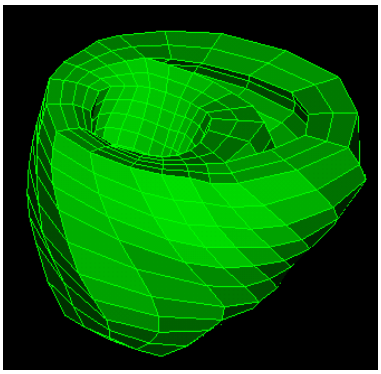
Patran is used as a finite element input file format and contains a multitude of data "cards", defining the properties of the mesh and boundary conditions. Each card is prepended by a number representing the type of data to follow. HDF5 is a binary file format commonly used in scientific computing. Data is stored in a hierarchical format similar to Linux/GNU operating system. A single HDF5 file is read by the PT in a manner similar to Patran files.

Meshing

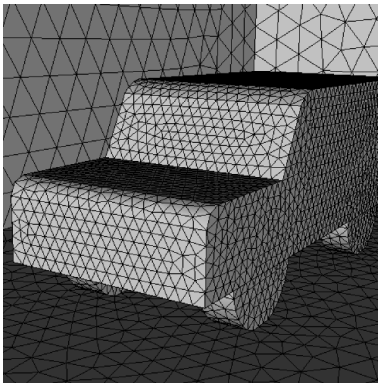
Meshing is primarily handled by the geometry and mesh generation toolkit Cubit developed by Sandia National Laboratory. Cubit is a lightweight CAD and finite element mesh generator capable of creating new and complex geometries from the bottom up as shown in figure 2.9.



(a) Cubi design used by the thermal module



(b) Mesh of the heart tissue used by the biomechanical module



(c) A truck model demonstrated more complex model

Figure 2.9: Example models and meshes as generated with Cubit

Cubit can generate triangle, quadrilateral, tetrahedral and hexahedral elements and export them with boundary conditions to the Patran mesh file. Cubit also has a built in

scripting language and journal system allowing for parametric mesh generation. Cubit is used to generate all meshes used by the modules presented in this paper [13].

Due to the limitation of using python based programs like Cubit, another mesh tool, Gmsh [15], is used. Gmsh is a mesh library developed by Christophe Geuzaine and Jean-François Remacle that can also be run as a standalone executable. Gmsh is used as a part of the meshing module to regenerate the internal mesh in the biomechanical case study.

The meshing relies on surface data and nodal connectivity provided by an input mesh file obtained from Cubit and the resultant updated nodal coordinates from the biomechanical module. The geometrical information is updated, and Gmsh is used to generate a new internal tetrahedral mesh which is then exported to a new Patran file used by the fluid module.

2.5.2 Post-Processing

Paraview

Paraview is an open source visualization tool accepting the Tecplot[16], VTK, and ASCII file formats. These output files are all generated by the thermal, fluid and biomechanical modules presented in this document. Figure 2.10 is an example output of contracting cardiac tissue. Other available tools used to visualize the simulated data are Tecplot360 and VisIt[17].

2.5.3 Solvers

Trilinos

Trilinos is a suite of object oriented numerical packages for building scientific applications. It is developed to run in parallel on high performance computers. Packages available in Trilinos include mesh generation and discretization, linear and non-linear solvers, numerical linear algebraic data handlers, and other scientific utilities. Most of these packages are developed independently, but are capable of interacting with other packages. The package Epetra is

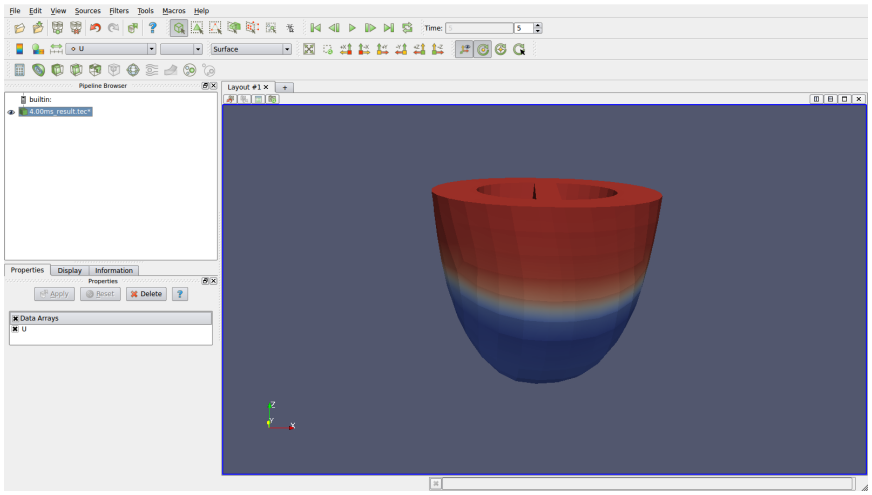


Figure 2.10: Paraview visualizing the heart model and the resultant stimulation and distortion

used to handle the distributed data structures and AztecOO is used to solve the system of equations.

Epetra is designed as an object oriented package that provides the foundation for other Trilinos to interact with. A number of built-in methods allow for manipulation of data in both serial and distributed parallel problems and acts as a wrapper for LAPACK and BLAS routines. Epetra also supports the generation and manipulation of dense matrices, sparse matrices. It includes implementations in C, C++ and Fortran.

AztecOO is an object oriented interface for the Krylov sparse matrix solver. The interface utilizes the vectors and matrices generated through Epetra and must use the Epetra_LinearProblem object. The AztecOO package comes with a suite of parallel iterative solvers such as CG, CGS, BiCGSTAB, GMRES or TFQMR, and a variety of preconditioners.

MAGMA

MAGMA is a specialized specialized dense matrix solver developed by the Innovative Computing Laboratory (ICL) at the University of Tennessee [10]. The solver aims to take advantage of high performance GPGPU's. MAGMA is capable of supporting systems utilizing OpenCL and Nvidia CUDA.

HIPS

HIPS is a Hierarchical Iterative Parallel Solver, similar to AztecOO, designed for sparse linear systems. Hips has API support for both C and Fortran programs. Unfortunately it is currently no longer under development.

Chapter 3

Loosely Coupled Physics Systems

Multiphysics systems are problems that combine multiple sets of governing principles and equations to solve real life problems. Some examples of multi-physics systems include, but are not limited to, thermal-fluid systems, fluid structure interaction, biomechanical processes, thermal stress analysis, piezoelectric simulations, and magnetohydrodynamics.

An example problem utilizing multiple physics systems is shown in figure 3.1. In this example three methods of conjugate heat transfer, conduction, radiation and convection, are combined with external fluid flow to form a multiphysics system.

In each of the previously mentioned example are instances of physics coupling with real world applications. This makes simulating multi-physics problems of great interest to the scientific community and society as a whole. The task in solving coupled systems of equations becomes quite daunting as the complexity and size of the problems grow.

A loosely coupled physics system is a set of physics equations that have been implemented in numerical solvers and designed to interact through a series of common geometrical boundaries. This allows one to separate the solution of the physics domains and require the only data exchange be the information on the shared boundary which makes the system "loosely" coupled. An advantage of this coupling method is the improvement of solver efficiency. This becomes most apparent with systems mixing all-to-all systems of equations, such as radiative

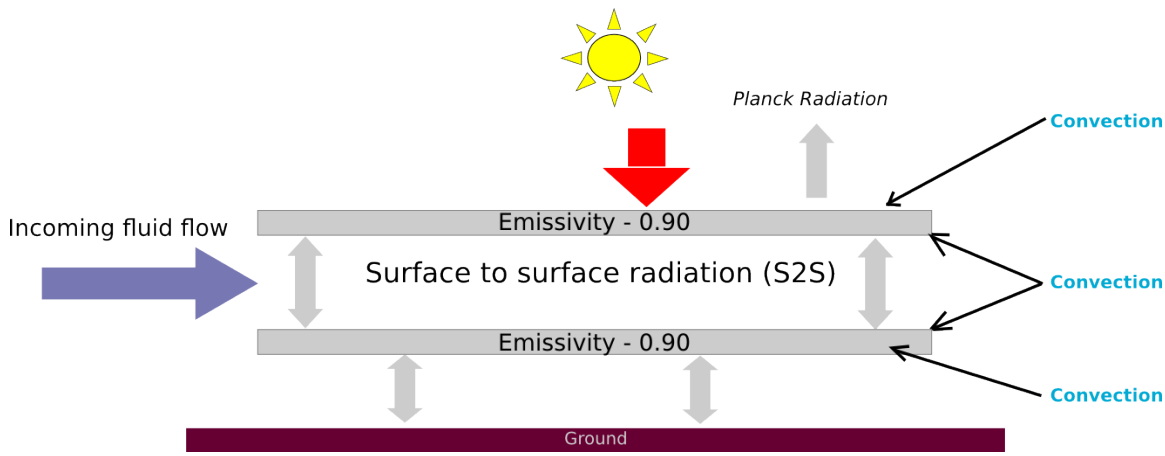


Figure 3.1: Multi physics system for conjugate heat transfer, radiation and fluid flow

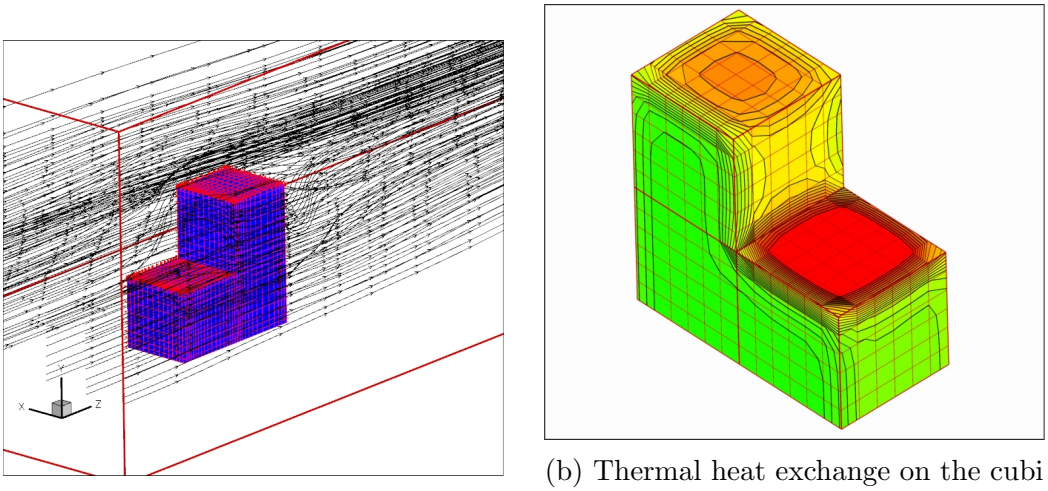
heat exchange, with localized systems of equations such as conduction.

The Interoperable Executive Library is designed to solver a set of loosely coupled systems each having its own components and definition.. This is particularly evident with the use of configuration files and the communicator library. The configuration file organizes the required physics modules and assembles them into a multiphysics system as well as defining the boundary points shared between the modules. In COMMLIB this coupling continues by providing functions for sending and receiving the data for the shared boundary conditions.

3.1 Thermal-Fluid Test Case

A thermal-fluid system is a multiphysics problem involving the principles of conservation of energy, mass, and momentum for solid and fluid domains. These problems have many real world applications in heat transfer studies ranging from industrial cooling systems to orbital reentry. For this test case the results of a model called "cubi" will be presented in figures 3.2a and 3.2b

In the cubi study the energy balance on a series of 2D planar surfaces in 3D Cartesian



(a) Cubi model in potential flow field

(b) Thermal heat exchange on the cubi model

Figure 3.2: Results obtained for the thermal-fluid test case on the cubi module

space will be analyzed. This type of simulation is particularly important because it combines three primary sources of heat transfer. The primary thermal module assembles the conservation of energy equations for the 2D object and imports the information for the boundaries from two other modules. These two modules handle the solution of the fluid flow field and the flux from radiative heat exchange. Also included in the thermal module is a function for incoming solar flux information provided by experimentation [18]. The solar flux calculation is a time dependent problem requiring the information about the position of the sun and the conditions of the field.

The cubi problem is used as an example in this thesis because of the computationally expensive nature of the physics involved and the relevance to real world applications. The radiation module requires the calculation of view factors, requiring a dense matrix solution for the radiosity terms. This part of the problem has been designed to utilize the MAGMA solver to offload these calculations to Graphics Processing Units (GPU's). The view factor calculations are generated through a pre-processor program called View3D [19]. The fluid

module also plays an important role in generating the convective terms.

The governing equation for the thermal module is Eq. 3.1 and is based on conservation of energy principles. Thermophysical properties are: ρ denotes the material density, C_p denotes the specific heat at constant pressure, k is the material conductivity, s is the thermal source term, and T is the dependent variable temperature.

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot [k(x)\nabla T] - s(x,t) = 0 \tag{3.1}$$

Boundary closure is covered in Eq. 3.2 in which the boundary conditions for radiation, convection, and flux terms are included. Properties in Eq. 3.2 are: h is the convection coefficient, σ is the Stefan-Boltzmann constant, ϵ is the emissivity, and $f(t)$ is the time based flux term.

$$k\nabla T \cdot n + h_{conv}(T - T_{atm}) + \sigma\epsilon(T^4 - T_{ref}^4) + f(t) \cdot n = 0 \tag{3.2}$$

Convective properties are calculated from the results of a potential flow solver. It gives the velocity distribution on the surface in the flow field, which is used to provide an estimate for the convective properties. The potential flow is derived from the conservation of mass giving a gradient of a scalar velocity potential ϕ, governed by Eq. 3.3.

$$\nabla^2 \phi = 0 \tag{3.3}$$

The general flow of the thermal-fluid test case is illustrated in figure 3.3. The simulation begins, as depicted, in the driver and is based on the input conditions of the configuration file. Execution then begins in the thermal module after reading data in from the PT. Throughout the execution of the simulation data is exchanged between the fluid and radiosity modules and the primary thermal module. During the simulation the thermal module writes the results to corresponding tecplot files.

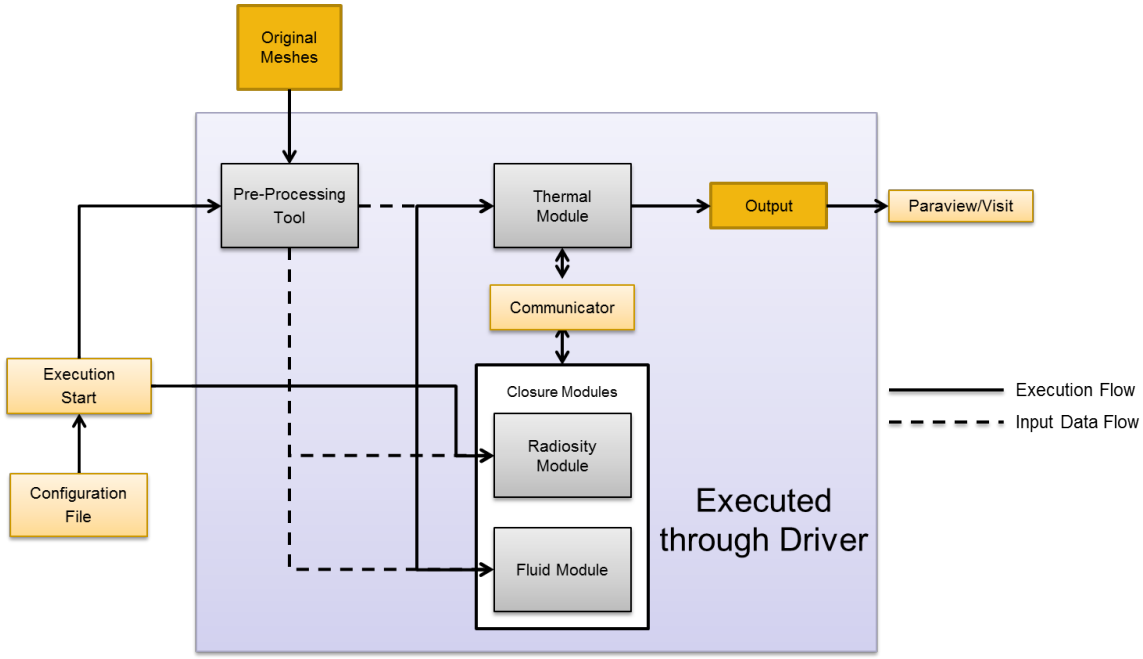


Figure 3.3: Computational workflow thermal-fluid system workflow using the fluid, radiosity and thermal modules in a multi-physics setting

3.2 Biomechanical Test Case

The biomechanical test case is a study of the interaction of stimulated and contracting cardiological tissue with an associated fluid flow. It is modeled by a monodomain reaction-diffusion equation representing the electrical impulses in the tissue and stress in the muscles. The internal fluid flow is then computed using an Incompressible Navier-Stokes solver.

This simulation will be focused on the ventricle section of the heart. In the top center section of the heart tissue is an area representing the atrioventricular node which acts as the point of initial stimulation for the entire system. As the action-potential propagates out the tissue contracts according to a given set of Kirchoff stress parameters and stress activated ion channels. Figure 3.4a gives an example of how this signal propagates and the resultant deformation in the tissue. As shown in figure 3.4b, the fluid domain is then solved with

the fluid module for the deformed grid. The simulation will not be using fully coupled fluid structure interaction but will instead have an updating interior flow profile as generated by a mesh module.

There are many reasons for performing such a simulation as a test of the IEL. By coupling a biomechanical problem with a fluid solver a great number of biological phenomena can be simulated and studied. This allows a researcher to gain a better understanding of not just the heart but any similar processes involving the interaction of fluid and electrical stimulations of the muscle tissue. As the models themselves become more comprehensive, parametric studies on drugs, mineral and vitamin concentrations can be conducted providing enormous insight into how biological processes function.

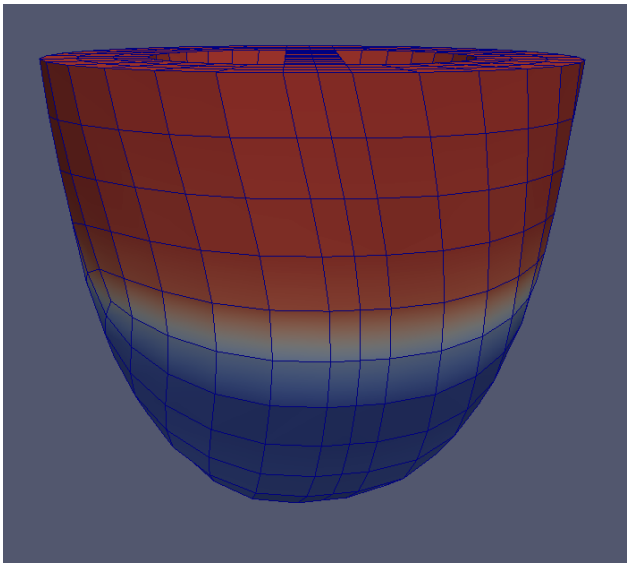
Using the IEL, this problem showcases the capability to integrate externally developed software and makes use of advanced scheduling techniques. The biomechanical module is a derivative of software developed at the University of Tennessee. It has been converted with minimal changes to work under the IEL.

Eqns. 3.4 and 3.5 represent the governing set of equations for the biomechanical module. Eq. 3.4 represents the mechanical portion of the solver, handling the stresses involved with the contraction of the tissue due to electrical stimulation. Eq. 3.5 is the reaction-diffusion equation and is the primary driving force of the simulation. In Chapter 6 this equation will be solved by breaking the PDE and the set of ODE's using a stepwise procedure.

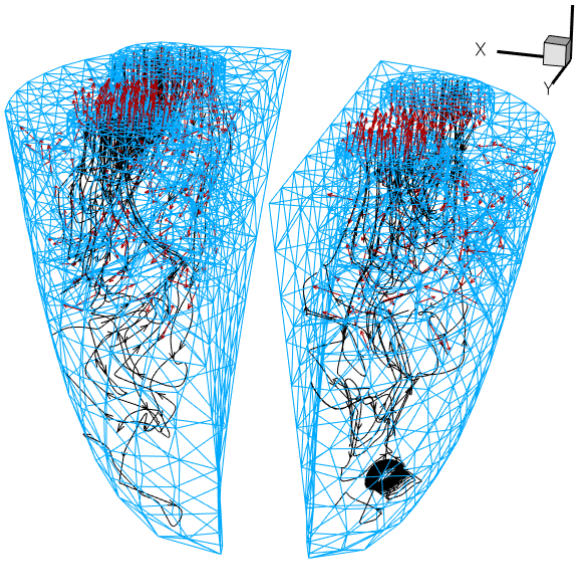
$$\nabla \cdot \sigma + b = 0 \tag{3.4}$$

$$\frac{\partial V}{\partial t} - \nabla \cdot (D \cdot \nabla V) + \frac{I_{ion}}{C} = 0 \tag{3.5}$$

Because the velocity of the internal flow is low, the flow field is incompressible in nature. The conservation of energy equation can be decoupled from the momentum equations. Eqns. 3.6 represents the conservation of mass and the 3.7 represents the momentum equations.



(a) Electro-Mechanical simulation of the heart tissue showing the action potential wave and the resultant deformation



(b) Internal flow field using the Incompressible Navier-Stokes solver

Figure 3.4: Simulation of the biomechanical system on the ventricles of a heart model

$$\nabla \cdot v = 0 \tag{3.6}$$

$$\rho \left(\frac{\partial v}{\partial t} + v \cdot \nabla v \right) = -\nabla p + \mu \nabla^2 v \tag{3.7}$$

As mentioned this simulation aims to demonstrate more advanced scheduling techniques using the driver. Figure 3.5 is an illustration of the workflow of the module. The simulation uses time integration in the driver during which execution of individual configuration files are performed. The time loop starts with the execution of the biomechanical module with the results saved in a restart and tecplot file.

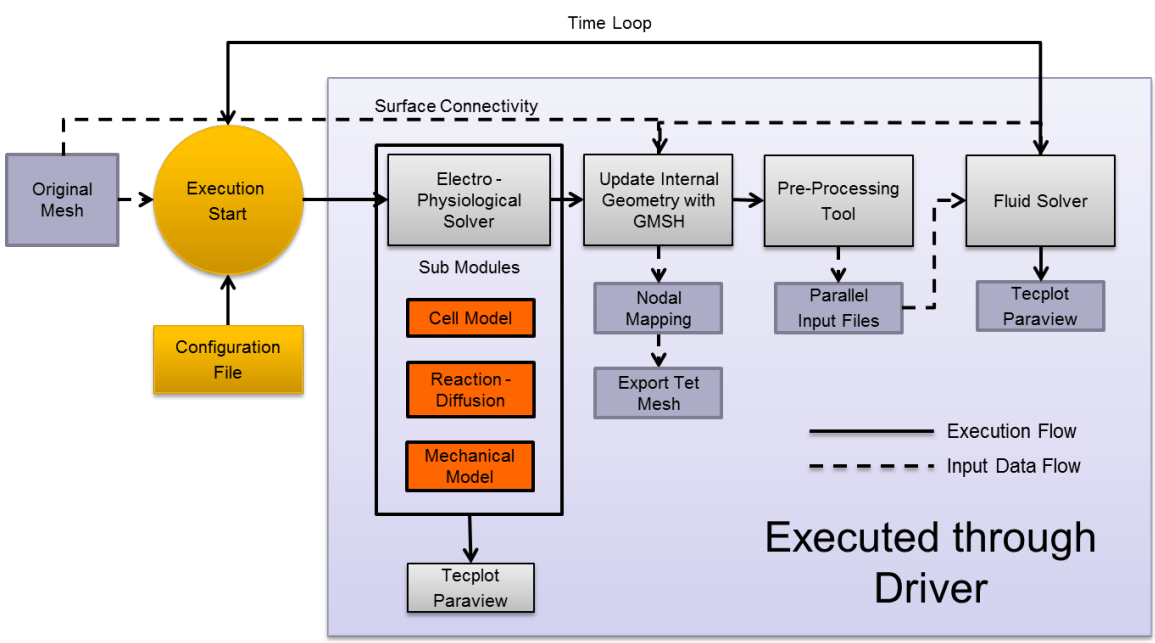


Figure 3.5: Workflow for the biomechanical test case using the biomechanical, mesh, pre-processing and fluid modules

After the biomechanical solver completes a geometry module regenerates the internal fluid domain using information from the original mesh and the recently written tecplot files.

Once the internal geometry is generated the PT then processes the new mesh for the fluid module. Once the end of the time loop is reached the driver increments the time step and continues till the end of the simulation.

3.3 Computational Implementation

The physics modules presented here utilize the finite element method. The finite element method was originally developed to solve structural problems using triangular elements. It was then further developed in the 1960's where it became a formalized method. One of the key features of this method is how the domain is divided into multiple subdomains fittingly called finite elements [20]. Combining these elemental submatrices an approximation to the solution of a problem is developed.

The finite element solution presented in this work utilizes a specific finite element template system that can be characterized as a problem solving environment. For most models the PDE can be rewritten as a series of matrix based templates which can be reused for problems of a similar nature [21].

Finite Element Method

The finite element formulation for a simple heat conduction problem will be rendered here as an example. This will demonstrate the steps going from the PDE, to the Galerkin Weak Statement and finally to the FE template.

The method starts off by defining the differential equation and the associated boundary conditions for the system. Eqns. 3.8 and 3.9 define this problem using *state variable* notation.

$$\mathcal{L}(T) = -\nabla \cdot (k(x)\nabla T) - s = 0 \quad on\ \Omega \tag{3.8}$$

$$T(x_a) = a; \quad T(x_b) = b; \quad on\ \partial\Omega \tag{3.9}$$

The finite element method aims to calculate an approximation of the physical domain. For this case the approximation is represented by Eq. 3.10, in which T^N is the approximate results and e^N is the associated error. This statement also allows one to determine, if necessary, the measure of the error upon substitution back into Eq. 3.8.

$$T(x) = T^N(x) + e^N(x) \tag{3.10}$$

Now the differential equation can be written as a function of $\mathcal{L}(T^N)$.

$$\mathcal{L}(T^N) = -\nabla \cdot (k(x)\nabla T^N) - s \neq 0 \quad on\, \Omega \tag{3.11}$$

In order to control the size of the error Eq. 3.11 is now written in the *weak form* by including the test space function Φ.

$$WS^N = \int_\Omega \Phi(x)\mathcal{L}(T^N)d\Omega \equiv 0 \quad on\, \Omega \tag{3.12}$$

Minimizing the error requires an optimal choice for ϕ which renders the error e^n *orthogonal* to the trial space. Called the *Galerkin Criterion* for a weak statement, this also implies the interpolation functions will be identical to the test space function. With the inclusion of the *Galerkin Criterion* integration-by-parts is then performed on Eq. 3.12 giving the following Galerkin Weak Statement.

$$GWS^N = \int_\Omega \nabla\Psi \cdot k\nabla T^N d\Omega - \int_\Omega \Psi s d\Omega - \int_{\partial\Omega} \Psi k \nabla T^N d\sigma \tag{3.13}$$

Eq. 3.14 defines the approximation of the solution with the interpolation function Ψ.

$$T^N(x) = \sum_{\alpha=1}^{N} \Psi(x) Q_\alpha \tag{3.14}$$

The finite element discrete implementation of Eq 3.13 requires replacing the trial and test space functions ϕ with $\{N_k\}$, the integral over Ω to an integration over the local element

Ω_e and then an *assembly* of each local elemental matrix. Therefore, the approximation in Eq. 3.14 can be replaced with

$$T_e(x) = \{N_k\}^T \{Q\}_e \tag{3.15}$$

Inserting Eq. 3.15 into Eq. 3.13 gives the discretized form of the Galerkin Weak Statement.

$$GWS^h = S_e \left[\int_{\Omega_e} \nabla\{N_k\} \cdot k_e \nabla\{N_k\}^T d\Sigma \{Q\}_e - \int_{\Omega_e} \{N_k\} s_e dx - \int_{\partial\Omega_e} \{N_k\} k \nabla\{N_k\}^T d\sigma \{Q\}_e = \{0\} \right] \tag{3.16}$$

The assembly operator, S_e, introduces the matrix-addition process associated with the finite element method. To better handle the element by element nature of finite elements an "Object-Oriented" method for representing the elemental matrices was developed for programming. This method breaks each portion of the GWS as follows.

$$\{WS\}_e = (const)\,[Matrix] \begin{Bmatrix} Q \quad or \\ \\ data \end{Bmatrix}_e \tag{3.17}$$

The subscript e represents the data which is element dependent. The brackets represents the following entities.

$$\begin{aligned} &(\cdot) \Rightarrow scalars \\ &\{\cdot\} \Rightarrow a\ column\ matrix\ of\ element\ DOF\ length \\ &[\cdot] \Rightarrow a\ squar\ matrix\ of\ element\ DOF\ order \end{aligned} \tag{3.18}$$

Of the template presented, the most important object is the $[Matrix]$. This object is written with the format $[Mbcccd]$. The M represents the matrix prefix with M can be

A, B, C, $n = 1, 2, 3$, and b is an integer for the number of FE bases ,$\{N_k\}$, in the component of the $\{WS\}_e$. The c's that follow then indicate the differentiation of the basis. Finally the d acts as a label for any specialized delineation.

Using this information for the FE Templates, Eqn. 3.16 can be represented as a series of templates as follows.

$$\begin{aligned}\{WS\}_e \equiv & \left([DIFF]_e + [BC]_e\right)\{Q\}_e - \{b\}_e \\ & () \, (COND) \, \{\} \, (0; -1) \, [M2KK] \, \{Q\} \\ & () \, () \, \{\} \, (0; 1) \, [M200] \, \{SRC\} \\ & () \, (COND) \, \{\} \, (?) \, [M20K] \, \{Q\}\end{aligned} \tag{3.19}$$

Newton-Raphson Method

For the majority of the simulations presented, a non-linear Newton-Raphson method is used to solve the system of equations at each time step. This method is derived from the basic Newton's method in Eq. 3.20. Here, the function f is the residual representation of the PDE in question, x is the value and m is the relaxation term.

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)} \tag{3.20}$$

Eq. 3.20 can be applied to a nonlinear system of equations by rearranging the system into Eq. 3.21. The function f can be replaced by any physics problem defined using the finite element method. Eq. 3.21 then takes on the form of Eq. 3.22 replacing the derivative term with the Jacobian.

$$f'(x_n)\,\delta x = -f(x_n) \tag{3.21}$$

$$[J]\{\delta Q\} = [-F_Q] \tag{3.22}$$

The Jacobian is a matrix of all first order partial derivatives of the the functions f. At each iteration of the Newton solver the change in the δQ is calculated until it reaches convergence criteria. Eq. 3.23 is an example system of four equations and four unknowns. On the left the Jacobian matrix is shown as a matrix of the derivatives, followed by the δ values and then the functions on the right.

$$\begin{bmatrix} \frac{\partial F_\phi}{\partial \phi} & \frac{\partial F_\phi}{\partial U} & \frac{\partial F_\phi}{\partial V} & \frac{\partial F_\phi}{\partial W} \\ \frac{\partial F_U}{\partial \phi} & \frac{\partial F_U}{\partial U} & \frac{\partial F_U}{\partial V} & \frac{\partial F_U}{\partial W} \\ \frac{\partial F_V}{\partial \phi} & \frac{\partial F_V}{\partial U} & \frac{\partial F_V}{\partial V} & \frac{\partial F_V}{\partial W} \\ \frac{\partial F_W}{\partial \phi} & \frac{\partial F_W}{\partial U} & \frac{\partial F_W}{\partial V} & \frac{\partial F_W}{\partial W} \end{bmatrix} \begin{Bmatrix} \Delta\phi \\ \Delta U \\ \Delta V \\ \Delta W \end{Bmatrix} = \begin{bmatrix} -F_\phi \\ -F_U \\ -F_V \\ -F_W \end{bmatrix} \tag{3.23}$$

Chapter 4

Fluid Modules

The fluid modules consists of solvers for potential and incompressible flow. The potential flow solver is used in the thermal-fluid system test case. It provide the elemental surface velocity which is used by the thermal module to calculate the convective properties. For the biomechanical test case the Incompressible Navier-Stokes solver generates the solution for the internal flow field as the geometry is updated during the simulation.

Both modules are derived from the same finite element framework written in C++. The modules are parallelized using MPI and make use of the Trilinos packages. Epetra is used to manage distributed vectors and matrices, and AztecOO to solve the sparse matrix.

4.1 Governing Equations

Potential Flow

Potential flow is one of the most basic classes of fluid mechanics problems and it is derived from the principles of conservation of mass. In the case of Eulerian fluid mechanics one can apply the Reynolds transport theorem to the differential statement for an incompressible fluid in a vanishing divergence of the velocity field u.

$$\mathcal{L}(v) = \nabla \cdot v = 0 \tag{4.1}$$

For steady, incompressible and irrotational flow, Eq. 4.2 holds.

$$curl\ v = \nabla \times v \equiv 0 \tag{4.2}$$

A scalar potential function ϕ, that also satisfies the irrotational flow for the velocity field, is then assigned to the velocity field v.

$$v \equiv \nabla \phi \tag{4.3}$$

Inserting ϕ into the previous equation gives the equation for potential flow.

$$\nabla^2 \phi = 0 \tag{4.4}$$

The velocity field is generated from the resultant gradient of ϕ. Eq. 4.5 defines this velocity field distribution in a compact tensor notation where n represents the total number of dimensions.

$$\frac{\partial \phi}{\partial x_i} = v_i, \quad 1 < i < n \tag{4.5}$$

Incompressible Flow

The Navier-Stokes equations provide the general description for the motion of fluid. These equations are defined from the principles of conservation for Mass, Momentum and Energy [22]. Equations 4.6 and 4.7 below represent the Incompressible Navier-Stokes system.

$$\nabla \cdot v = 0 \tag{4.6}$$

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \mu \nabla^2 v \tag{4.7}$$

To satisfy the incompressibility constraint many possibilities arise to handle the enforcement of conservation of mass principles. Several algorithms have been developed over the years including, but not limited to, the SIMPLE and PISO algorithms. The Continuity Constraint Method (CCM) developed by Williams [23] is used in this project.

The CCM utilizes a multi step process of evaluating an initial velocity field u^*, solving for a pseudo-pressure "C", and subsequently reaching the final velocity field. At each timestep of the solution this process is repeated until the non-linear system reaches a specified convergence criteria. The strategy for the CCM iterative solution is:

(1)	Initialize constraint variable $C_{p+1}^{n+1} = C^n$
(2)	Solve the momentum equations solving for u^{n+1}
(3)	Solve Poisson's equation for ϕ
(4)	Update approximation for C with ϕ
(5)	Repeat steps (2)-(4) until convergence
(6)	Advance time step

Applying the CCM to Eq. 4.7 and introducing the Poisson's equation for ϕ one arrives at equations 4.8 and 4.9. The given pressure, P, has been replaced with a pseudo-pressure C and can be recovered from the solution of the velocity field.

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla C + \mu \nabla^2 v \tag{4.8}$$

$$\nabla^2\phi = \nabla \cdot v \tag{4.9}$$

4.2 Finite Element Implementation

Potential Flow

The Potential flow system is defined in the following equations.

$$\mathcal{L}(\phi) = \nabla^2\phi = 0 \quad on\, \Omega \tag{4.10}$$

$$\mathcal{L}(\phi, v_i) = \frac{\partial\phi}{\partial x_i} - v_i = 0 \quad 1 < i < n \tag{4.11}$$

$$\ell(\phi) = \nabla\phi \cdot \hat{n} = \vec{v}_{inlet} \cdot \vec{n} \quad on\, \partial\Omega_1, inlet \tag{4.12}$$

$$\ell(\phi) = \nabla\phi \cdot \hat{n} = 0 \quad on\, \partial\Omega_2, wall \tag{4.13}$$

$$\ell(\phi) = \phi = 0 \quad on\, \partial\Omega_3, outlet \tag{4.14}$$

Using weighted residuals, the weak formulation is

$$\begin{aligned} WS_1^N &= \int_\Omega \Psi(\tau)\mathcal{L}(\phi^N)d\tau \equiv 0 \\ WS_2^N &= \int_\Omega \Psi(\tau)\mathcal{L}(v_i^N, \phi^N)d\tau \equiv 0 \end{aligned} \tag{4.15}$$

Substituting Eqns. 4.10 - 4.14 into Eq. 4.15 gives

$$\begin{aligned} WS_1^N &= \int_\Omega \Psi(\tau)\nabla^2\phi^N(\tau)d\tau \equiv 0 \\ WS_2^N &= \int_\Omega \Psi(\tau)(\frac{\partial\phi^N}{\partial x_i} - v_i^N) \equiv 0 \end{aligned} \tag{4.16}$$

After integrating the weak statement, applying Greene-Gauss's Theorem, and assuming that the test function is equal to the interpolation function, the system reaches an optimal state through the *Galerkin Criterion*.

$$
\begin{aligned}
GWS_1^N &= \int_\Omega \nabla\Psi \cdot \nabla\phi^N d\tau - \oint_{\partial\Omega} \Psi \nabla\phi^N \cdot n d\sigma \equiv 0 \\
GWS_2^N &= \int_\Omega \Psi \frac{\partial \phi^N}{\partial x_i} d\tau - \int_\Omega \Psi V_{i,\alpha}^N d\tau \equiv 0
\end{aligned}
\tag{4.17}
$$

Applying boundary conditions from Eqns. 4.12 and 4.13 to 4.17 gives 4.18.

$$
GWS_1^N = \int_\Omega \nabla\Psi \cdot \nabla\phi^N d\tau - \oint_{\partial\Omega} \Psi \frac{\partial \phi^N}{\partial n} d\sigma \equiv 0 \tag{4.18}
$$

Using the set of templates as discussed in Section 3.3 the system of equations can be rewritten as follows

$$
\begin{aligned}
F_\phi &= [C2KK]\{\phi\} + [Cn200]\{\vec{V} \cdot n\} \\
F_U &= [C20X]\{\phi\} - [C200]\{U\} \\
F_V &= [C20Y]\{\phi\} - [C200]\{V\} \\
F_W &= [C20Z]\{\phi\} - [C200]\{W\}
\end{aligned}
\tag{4.19}
$$

Taking the functions derived in Eq. 4.19 the system of equations for the Newton solver then becomes,

$$\begin{bmatrix} [C2kk] & 0 & 0 & 0 \\ [C20X] & [C200] & 0 & 0 \\ [C20Y] & 0 & [C200] & 0 \\ [C20Z] & 0 & 0 & [C200] \end{bmatrix} \begin{Bmatrix} \Delta\phi \\ \Delta U \\ \Delta V \\ \Delta W \end{Bmatrix} = \begin{bmatrix} -\left([C2kk]\{\phi\} + [C200]\left\{\vec{V}\cdot n\right\}\right) \\ -\left([C20X]\{\phi\} + [C200]\{U\}\right) \\ -\left([C20Y]\{\phi\} + [C200]\{V\}\right) \\ -\left([C20Z]\{\phi\} + [C200]\{W\}\right) \end{bmatrix} \tag{4.20}$$

Because the solution to the potential flow is a linear problem the Newton non-linear solver used in the module will converge after one iteration.

Incompressible Flow

Momentum Equations

The finite element formulation for the momentum equation begins by expanding across each of the direction fields and then non-dimensionalizing the system. This introduces the Reynolds number, Re, which is the ratio of inertial to viscous forces.

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} &= -\frac{\partial C}{\partial x} + \frac{1}{Re}\left(\nabla^2 u\right) \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} &= -\frac{\partial C}{\partial y} + \frac{1}{Re}\left(\nabla^2 v\right) \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} &= -\frac{\partial C}{\partial z} + \frac{1}{Re}\left(\nabla^2 w\right) \end{aligned} \tag{4.21}$$

Introducing a second order time accurate Taylor polynomial given in Eq. 4.22, the Taylor polynomial takes into account the explicit, implicit or mixed numerical schemes using the θ term.

$$\bar{U}^{n+1} = \bar{U}^{n} + \left[(1-\theta) \frac{\partial \bar{U}^{n}}{\partial t} \Delta t + \theta \frac{\partial \bar{U}^{n+1}}{\partial t} \Delta t \right] + \mathcal{O}\left(\Delta t^2\right) \tag{4.22}$$

Assuming an implicit scheme with $\theta = 1$ Eq. 4.22 becomes the backward Euler scheme

$$\bar{U}^{n+1} = \bar{U}^{n} + \theta \frac{\partial \bar{U}^{n+1}}{\partial t} \Delta t + \mathcal{O}\left(\Delta t^2\right) \tag{4.23}$$

Substituting the momentum equations Eq. 4.21 into Eq. 4.23 converts the system of PDE's to *state variable* notation.

$$\begin{aligned}
\mathcal{L}\left(u^{n+1}\right) &= \left(u^{n+1} - u^{n}\right) + \Delta t \left[u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{\partial C}{\partial x} - \frac{1}{Re}\left(\nabla^2 u\right) \right]^{n+1} = 0 \\
\mathcal{L}\left(v^{n+1}\right) &= \left(v^{n+1} - v^{n}\right) + \Delta t \left[u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{\partial C}{\partial y} - \frac{1}{Re}\left(\nabla^2 v\right) \right]^{n+1} = 0 \\
\mathcal{L}\left(w^{n+1}\right) &= \left(w^{n+1} - w^{n}\right) + \Delta t \left[u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{\partial C}{\partial z} - \frac{1}{Re}\left(\nabla^2 w\right) \right]^{n+1} = 0
\end{aligned} \tag{4.24}$$

Converting to the weak statement requires integration over the domain and multiplication with the test space function Ψ.

$$\begin{aligned}
WS_U &= \int_{\Omega} \Psi \mathcal{L}\left(u^{n+1}\right) d\Omega = 0 \\
WS_V &= \int_{\Omega} \Psi \mathcal{L}\left(v^{n+1}\right) d\Omega = 0 \\
WS_W &= \int_{\Omega} \Psi \mathcal{L}\left(w^{n+1}\right) d\Omega = 0
\end{aligned} \tag{4.25}$$

Expanding the integrals and performing integration by parts gives the following set of weak statements coupled with natural boundary conditions.

$$
\begin{aligned}
WS_U = \int_\Omega \Psi \left(u^{n+1} - u^n\right) d\Omega + \Delta t \int_\Omega \Psi \left[u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{\partial C}{\partial x}\right] d\Omega \\
+ \Delta t \frac{1}{Re} \int_\Omega \nabla\Psi \cdot \nabla u d\Omega - \Delta t \int_{\partial\Omega} \Psi \frac{\partial u}{\partial n} d\sigma = 0 \\
WS_V = \int_\Omega \Psi \left(v^{n+1} - v^n\right) d\Omega + \Delta t \int_\Omega \Psi \left[u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{\partial C}{\partial y}\right] d\Omega \\
+ \Delta t \frac{1}{Re} \int_\Omega \nabla\Psi \cdot \nabla v d\Omega - \Delta t \int_{\partial\Omega} \Psi \frac{\partial v}{\partial n} d\sigma = 0 \\
WS_W = \int_\Omega \Psi \left(w^{n+1} - w^n\right) d\Omega + \Delta t \int_\Omega \Psi \left[u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{\partial C}{\partial z}\right] d\Omega \\
+ \Delta t \frac{1}{Re} \int_\Omega \nabla\Psi \cdot \nabla w d\Omega - \Delta t \int_{\partial\Omega} \Psi \frac{\partial w}{\partial n} d\sigma = 0
\end{aligned}
\tag{4.26}
$$

Seeking to minimize the associated error one assumes the test function is equal to the interpolation functions, applying this brings Eq. 4.26 to the Galerkin Weak form.

$$
\begin{aligned}
GWS_U = &\int_\Omega \{N_k\}\{N_k\}^T \left(\{u^{n+1}\} - \{u^n\}\right) d\Omega + \Delta t \Bigg[\Bigg(\int_\Omega \{u^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial x} \{u^{n+1}\} d\Omega \\
&+ \int_\Omega \{v^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial y} \{u^{n+1}\} d\Omega + \int_\Omega \{w^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial z} \{u^{n+1}\} d\Omega \Bigg) \\
&+ \int_\Omega \{N_k\} \frac{\partial \{N_k\}^T}{\partial x} \{C\} d\Omega + \frac{1}{Re} \int_\Omega \nabla\{N_k\} \cdot \nabla\{N_k\}^T \{u^{n+1}\} d\Omega \\
&- \int_{\partial\Omega} \{N_k\} \frac{\partial \{N_k\}^T}{\partial n} \{u^{n+1}\} d\sigma \Bigg] = 0 \\
GWS_V = &\int_\Omega \{N_k\}\{N_k\}^T \left(\{v^{n+1}\} - \{v^n\}\right) d\Omega + \Delta t \Bigg[\Bigg(\int_\Omega \{u^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial x} \{v^{n+1}\} d\Omega \\
&+ \int_\Omega \{v^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial y} \{v^{n+1}\} d\Omega + \int_\Omega \{w^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial z} \{v^{n+1}\} d\Omega \Bigg) \\
&+ \int_\Omega \{N_k\} \frac{\partial \{N_k\}^T}{\partial y} \{C\} d\Omega + \frac{1}{Re} \int_\Omega \nabla\{N_k\} \cdot \nabla\{N_k\}^T \{v^{n+1}\} d\Omega \\
&- \int_{\partial\Omega} \{N_k\} \frac{\partial \{N_k\}^T}{\partial n} \{v^{n+1}\} d\sigma \Bigg] = 0 \\
GWS_W = &\int_\Omega \{N_k\}\{N_k\}^T \left(\{w^{n+1}\} - \{w^n\}\right) d\Omega + \Delta t \Bigg[\Bigg(\int_\Omega \{u^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial x} \{w^{n+1}\} d\Omega \\
&+ \int_\Omega \{v^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial y} \{w^{n+1}\} d\Omega + \int_\Omega \{w^{n+1}\}\{N_k\}\{N_k\} \frac{\partial \{N_k\}^T}{\partial z} \{w^{n+1}\} d\Omega \Bigg) \\
&+ \int_\Omega \{N_k\} \frac{\partial \{N_k\}^T}{\partial z} \{C\} d\Omega + \frac{1}{Re} \int_\Omega \nabla\{N_k\} \cdot \nabla\{N_k\}^T \{w^{n+1}\} d\Omega \\
&- \int_{\partial\Omega} \{N_k\} \frac{\partial \{N_k\}^T}{\partial n} \{w^{n+1}\} d\sigma \Bigg] = 0
\end{aligned}
\tag{4.27}
$$

As described in the Section 3.3 the finite element template form is used for computational implementation. In order to handle the solution of non-linear system of equations, a multivariate Newton-Raphson iterative method is implemented. This requires each of the momentum equations to be defined as a residual function F.

$$
\begin{aligned}
F_u = WS_{U,e} = & [C200]\left(\{u^{n+1}\} - \{u^n\}\right) + \Delta t\Bigg[\Big(\{u^{n+1}\}[C300X] + \{v^{n+1}\}[C300Y] + \\
& \{w^{n+1}\}[C300Z]\Big)\{u^{n+1}\} \\
& + [C20X]\{C\} + \frac{1}{Re}[C2KK]\{u^{n+1}\} - [Cn200]\{u^{n+1}\cdot n\}\Bigg] = 0 \\
F_v = WS_{V,e} = & [C200]\left(\{v^{n+1}\} - \{v^n\}\right) + \Delta t\Bigg[\Big(\{u^{n+1}\}[C300X] + \{v^{n+1}\}[C300Y] + \\
& \{w^{n+1}\}[C300Z]\Big)\{v^{n+1}\} \\
& + [C20Y]\{C\} + \frac{1}{Re}[C2KK]\{v^{n+1}\} - [Cn200]\{v^{n+1}\cdot n\}\Bigg] = 0 \\
F_w = WS_{W,e} = & [C200]\left(\{w^{n+1}\} - \{w^n\}\right) + \Delta t\Bigg[\Big(\{u^{n+1}\}[C300X] + \{v^{n+1}\}[C300Y] + \\
& \{w^{n+1}\}[C300Z]\Big)\{w^{n+1}\} \\
& + [C20Z]\{C\} + \frac{1}{Re}[C2KK]\{w^{n+1}\} - [Cn200]\{w^{n+1}\cdot n\}\Bigg] = 0
\end{aligned}
\tag{4.28}
$$

Taking the derivatives of the residual functions gives the following terms for the Jacobian matrix.

$$
\begin{aligned}
\frac{\partial F_u}{\partial u} &= [C200] + \Delta t[\{u^{n+1}\}[C300X] + \{v^{n+1}\}[C300Y] + \{w^{n+1}\}[C300Z] + \\
&\{u^{n+1}\}[C3X00] + \frac{1}{Re}[C2KK]] \\
\frac{\partial F_u}{\partial v} &= \Delta t\{v^{n+1}\}[C3Y00] \\
\frac{\partial F_u}{\partial w} &= \Delta t\{w^{n+1}\}[C3Z00]
\end{aligned}
\tag{4.29}
$$

$$
\begin{aligned}
&\frac{\partial F_v}{\partial v} = [C200] + \Delta t[\{u^{n+1}\}\,[C300X] + \{v^{n+1}\}\,[C300Y] + \{w^{n+1}\}\,[C300Z] + \\
&\{v^{n+1}\}\,[C3Y00] + \frac{1}{Re}\,[C2KK]] \\
&\frac{\partial F_v}{\partial u} = \Delta t\{u^{n+1}\}\,[C3X00] \\
&\frac{\partial F_v}{\partial w} = \Delta t\{w^{n+1}\}\,[C3Z00]
\end{aligned}
\tag{4.30}
$$

$$
\begin{aligned}
&\frac{\partial F_w}{\partial w} = [C200] + \Delta t[\{u^{n+1}\}\,[C300X] + \{v^{n+1}\}\,[C300Y] + \{w^{n+1}\}\,[C300Z] + \\
&\{w^{n+1}\}\,[C3Z00] + \frac{1}{Re}\,[C2KK]] \\
&\frac{\partial F_w}{\partial v} = \Delta t\{v^{n+1}\}\,[C3Y00] \\
&\frac{\partial F_w}{\partial u} = \Delta t\{u^{n+1}\}\,[C3X00]
\end{aligned}
\tag{4.31}
$$

Continuity Constraint

The continuity constraint method employs a pseudo-pressure variable, ϕ, as a mathematical solution to mitigate the stability problems associated with the Incompressible Navier-Stokes system. At each step of the non-linear solver, a solution to Eq. 4.32 is used to generate a value for the scalar function ϕ from the results of the update velocity field. The result is then used to update the continuity constraint C at each iteration of the Newton solver.

$$
\nabla^2\phi = \nabla \cdot v \tag{4.32}
$$

In the state variable form, the system becomes

$$
\mathcal{L}(\phi) = \nabla^2\phi - \nabla \cdot v = 0 \tag{4.33}
$$

After performing integration by parts one arrives at the weak statement.

$$WS = \int_{\Omega} \nabla\Psi \cdot \nabla\phi d\Omega + \int_{\Omega} \Psi\nabla \cdot v d\Omega - \oint_{\partial\Omega} \Psi \frac{\partial\phi}{\partial n} d\sigma = 0 \tag{4.34}$$

Following finite element templates Eq. 4.34 can now be written in the template form. The only boundary condition $\phi = 0$ is imposed on the outlet of the system.

$$F_{\phi} = [C2KK]\{\phi\} - ([C2OX]\{u\} + [C2OY]\{v\} + [C2OZ]\{w\}) = 0 \tag{4.35}$$

After calculating the solution ϕ the value for "C" can be updated via Eq. 4.36.

$$C^{i+1} = C^{i} + \frac{\phi}{\Delta t} \tag{4.36}$$

Eq. 4.35 is then solved by a pseudo non-linear solver.

$$\left[\frac{\partial F_{\phi}}{\partial \phi}\right]\{\Delta\phi\} = [-F_{\phi}] \tag{4.37}$$

4.3 Module Structure

This module is designed to be manipulated into solving a number of physics systems as using triangle, quadrilateral, tetrahedral or hexahedral meshes. Conversion for different physics problems requires minimal changes to the overall structure of the code with most changes occurring in the functions that assemble the right and left hand sides. This flexibility allows for reuse of the code for other finite element problems.

The flow of the module is illustrated in figure 4.1 The module begins its execution by reading the configuration files and initializing the data structures. These data structures consist of a number of C++ objects for specific data sets including, but not limited to, geometry information, solver configuration and boundary conditions. Once the data structures have been initialized, the boundary conditions, initial conditions and problem characteristics of the system of equations are fully described.

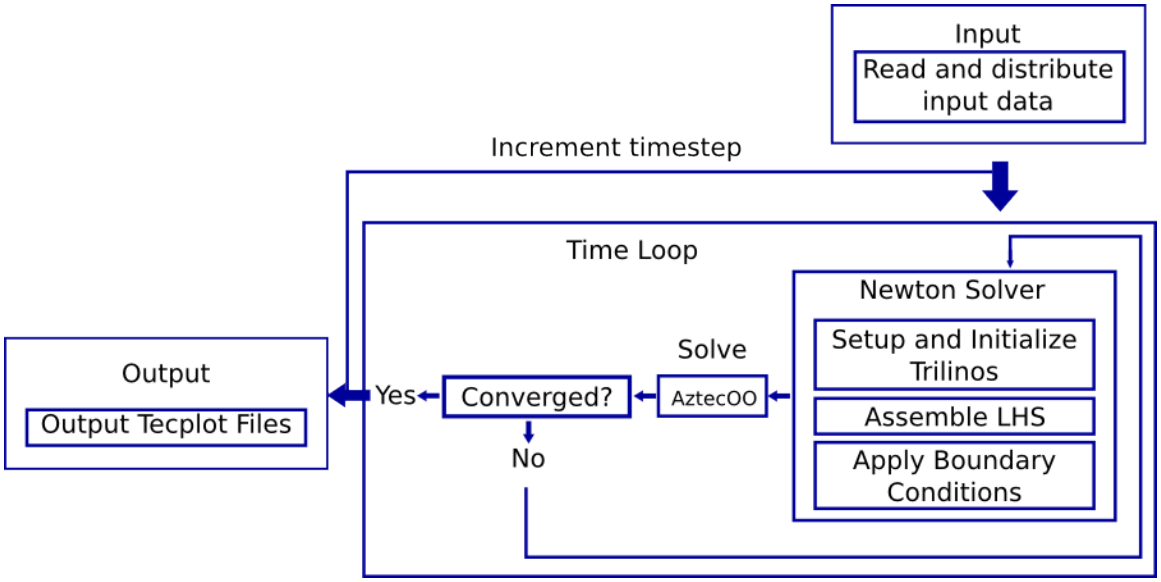


Figure 4.1: Overview of the structure and flow of the fluid modules

Inside the time loop is the Newton non-linear solver. This utilizes the solver method as defined in Section 3.3 and must iterate over again until the solution of the velocity field reaches a converged state. In the Newton loop the Epetra data structures are generated and filled using the assembly method as provided by the finite element method. Once the boundary conditions have been applied the AztecOO solver is called. If the changes of the residual of the state variables meet the convergence criteria the data is written to a tecplot file.

Chapter 5

Thermal Module

Conjugate heat transfer is handled through the thermal module and the radiosity module. Both modules are written in C. They utilize the AztecOO sparse matrix solver for conservation of energy equation and the MAGMA GPU solver for the solution of the radiation exchange. Other components include view factor calculation with View3D and the PT.

The thermal module is designed to simulate conjugate heat transfer on 2D surfaces in 3D space. The reasoning behind this type of simulation is to reduce the complexity of the system and allow for generation of very complex structures such as vehicles and buildings.

5.1 Governing Equations

The thermal module uses conservation of energy principles for conduction, convection and radiation. Eq. 5.1 defines the energy principle and Eq. 5.2 provides the boundary closure to the system.

$$\rho C_p \frac{\partial T}{\partial t} - \nabla \cdot [k(x)\nabla T] - s(x,t) = 0 \tag{5.1}$$

$$k\nabla T \cdot n + h_{conv}(T - T_{atm}) + \sigma\epsilon(T^4 - T_{ref}^4) + f(t) \cdot n = 0 \tag{5.2}$$

Where ρ, C_p, k,ϵ, σ denote density, specific heat, conductivity, thermal emissivity and Stefan-Boltzmann constant respectively. For the purpose of the work performed in this thesis, these properties will be considered constant.

The convective terms in Eq. 5.12 are handled through a series of correlations. These correlations depend on the surface velocity distribution and are calculated by the fluid module simulations for potential flow. The coefficient of convection is a combination of forced and natural convection depending on the Grashof, Prandtl and Reynolds numbers. The Grashof number, $Gr = \frac{\rho^2 \beta_{air} g \Delta T L_e^3}{\mu^2}$, represents the ratio of the buoyancy to viscous forces acting on a fluid. The Prandtl number represents the ratio of momentum to thermal diffusivity in a system, determined by $Pr = \frac{\mu C_{p,air}}{k_{air}}$. The final term is the Reynolds number, $Re = \frac{\rho U_x}{\mu}$, or the ratio of inertial forces to viscous forces.

Eqns. 5.3, 5.4 and 5.5 give the relationships for the convective terms. The overall convection coefficient is a mixture of both natural and forced convection.

$$h_{ccnv} = h_{not} \cdot \eta_{air} + h_{for} \tag{5.3}$$

$$h_{for} = C_1 k Pr^{\frac{1}{3}} Re^{\frac{1}{2}} L^{-1} \tag{5.4}$$

$$h_{nat} = C_2 k (GrPr)^n L^{-1} \tag{5.5}$$

The ratio of natural to forced convection is determined by the η term in Eq. 5.3. This switching term is calculated via Eq. 5.6. η is dependent upon the Grashof and Reynolds numbers, and therefore dependent of the surface velocity of the element.

$$\eta_{air} = \frac{Log(1 + \frac{Gr}{Re^2})}{1 + Log(1 + \frac{Gr}{Re^2})} \tag{5.6}$$

The radiosity formulation for radiative heat exchange is formulated depending on differ-

ential areas and therefore eliminates the coupling with Eq. 5.12. This formulation depends on the calculation of a *view factor*. The view factor, Eq. 5.7, is the portion of radiation emitted from a particular element that then impinges on another surface [24]. This view factor makes the solution of the radiation terms of all-to-all type, meaning each surface has a reaction with every other surface.

$$F_{k\to i} = \frac{1}{A_k}\int_{A_k}\int_{A_i}\frac{cos(\phi_k)cos(\phi_i)dA_k dA_i}{\pi r_{k\to i}^2} \tag{5.7}$$

The "kernel" of Eq. 5.7 is the portion of the integrals that display symmetry. Eq. 5.8 is the kernel in this case, where $K_{k\to i} = K_{i\to k}$ for all surfaces.

$$K_{k\to i} = \frac{cos(\phi_k)cos(\phi_i)dA_k dA_i}{\pi r_{k\to i}^2} \tag{5.8}$$

The total efflux on surface A_i is defined as the difference between the radiosity R_i and the sum of the incident radiation exchange from all other surfaces[18]. This simplifies to Eq. 5.9.

$$q_i = \frac{\epsilon_i}{\rho_i}\left[\sigma T_i^4 - R_i\right] \tag{5.9}$$

5.2 Finite Element Implementation

The finite element formulation follows the steps used in Section 3.3. It begins by assuming an approximation for the temperature distribution as defined by Eq. 5.10.

$$T^N(x,t) \equiv \sum_{e=1}^{N}\Phi(x)T(t) \tag{5.10}$$

By applying the *Galerkin Criterion* an optimized approximation is reached where the integral is set to zero.

$$GWS^N \equiv \int_\Omega \Psi(x)\mathcal{L}(T^N)d\tau \equiv 0 \tag{5.11}$$

Expanding Eq. 5.11 by substituting in 5.1 gives

$$\begin{aligned} GWS^N \equiv & \int_\Omega \Psi \left(\rho c_p \frac{\partial T^N}{\partial t} - s \right) d\Omega \\ & + k \int_\Omega \nabla\Psi \cdot \nabla T^N d\Omega \\ & + \oint_{\partial\Omega_{conv}\cap\partial\Omega} \Psi \left[h_{conv} \left(T^N - T_{amb} \right) \right] d\sigma \\ & + \oint_{\partial\Omega_{flux}\cap\partial\Omega} \Psi f(t) \cdot n \quad d\sigma - \oint_{\partial\Omega_{else}\cap\partial\Omega} \Psi k \nabla T^N \cdot n \quad d\sigma \end{aligned} \tag{5.12}$$

The final template form for the thermal module can be found in Appendix C.

The finite element formulation for the radiosity module follows the same procedure as the thermal module giving the Galerkin Weak Statement Eq. 5.13.

$$\begin{aligned} \{GWS\}_{e=i} = & \int_{\Omega_i} \{N\}_i \Bigg[\left(\{N\}^T \{R_i\} - \epsilon_i \sigma \{N\}^T \{T_i^4\} \right) dA_i \\ & - \rho_i \left(\sum_{k=1}^{n} \int_{\Omega_k} K_{i\to k} \{N\}^T \{R_k\} dA_i \right) dA_i \Bigg] = \{0\} \end{aligned} \tag{5.13}$$

Eq. 5.14 brings us to the final FEM template for the radiation terms.

$$\frac{det_i}{\rho_i} [Cn200] \{R_i\} - \frac{\epsilon_i \sigma det_i}{\rho_i} [Cn200] \{T^4\} - \sum_{k=1}^{n} K_{i\to k} det_k det_i [Cn1010] \{R_k\} = \{0\} \tag{5.14}$$

Unlike the assembly process as described in Chapter 4 the radiosity formulation follows a *hypermatrix* assembly. Each elemental value of Eq. 5.14 assembles in non-overlapping fashion in the solution matrix. The solution of the radiosity R_k is based on elementwise calculations.

5.3 Module Structure

The structure of the thermal module is very similar to that of the fluid modules in Chapter 4 as seen in figure 5.1. The PT is used to initialize the input data for parallel computation and the data files are read at the start of the module. The data structures and problem characteristics are defined before entering the time integration loop.

In the time integration loop the module executes the non-linear Newton solver. Within the Newton solver the system is formed from the finite element implementation in Eqns. C.1 and C.2. During this process the information of the boundary conditions is imported using the executive from the radiosity and fluid modules.

Once the system of equations have been assembled, AztecOO solves the system of equations. If the convergence criteria of the non-linear solver has been met the time step is incremented and the data is written to the tecplot files.

5.4 Test Case Results and Discussion

The results for the thermal-fluid system are illustrated in the following set of figures. The results for the potential flow solution are shown in figure 5.2 for the cubi model. The figures proceeding this potential flow solution show the change in surface temperature of the cubi module through a portion of the simulation. The details for the driver and configuration file can be found in Appendix B.1.

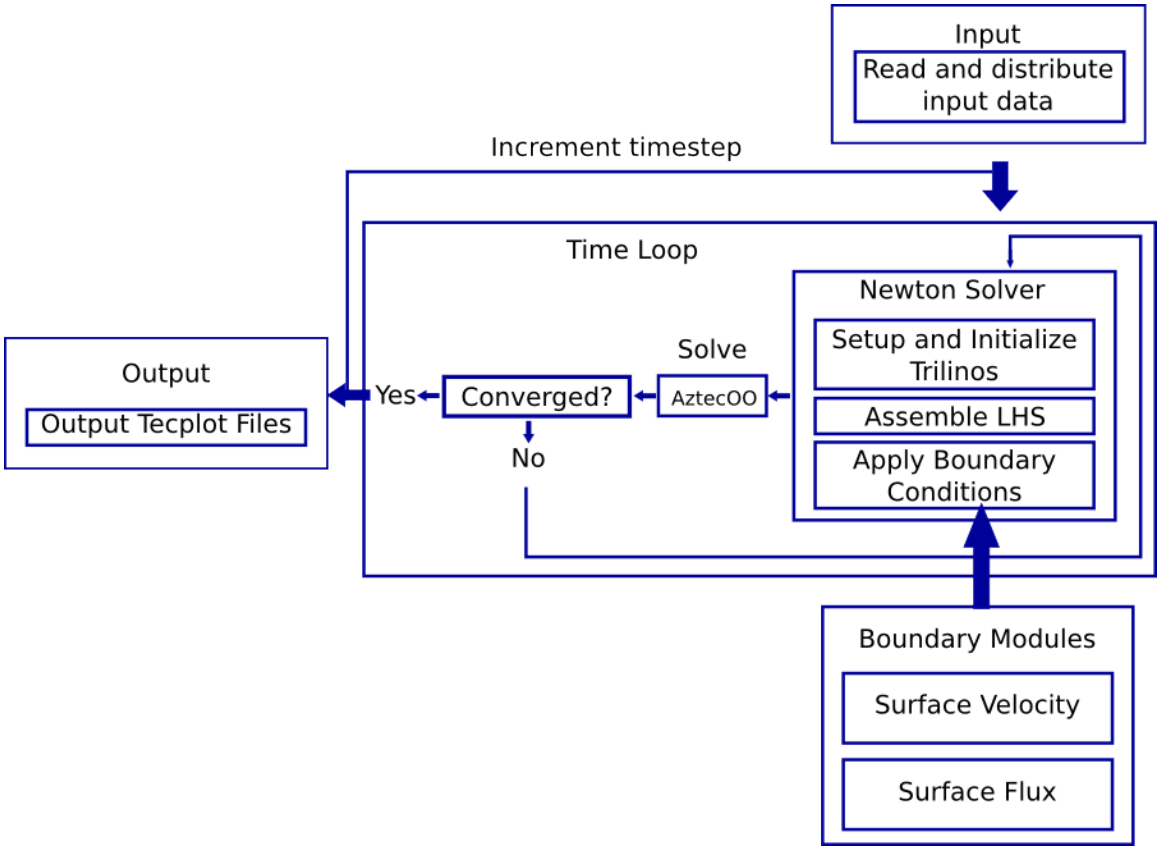


Figure 5.1: Thermal module structure and flow including shared boundary modules

Looking at figures 5.3 through 5.5 the effects of radiation exchange and solar flux on the surfaces of the model can be seen. This is most evident on the surfaces on the two surface sitting at 90 degree angles. These surfaces maintain a higher temperature due to the radiation exchange. Convection is not so evident in this case.

These results showcase the ability of the IEL to couple multiple separate physics modules execute them in a multiphysics simulation. Also shown is the capability to exchange shared boundary information between multiple modules and processes.

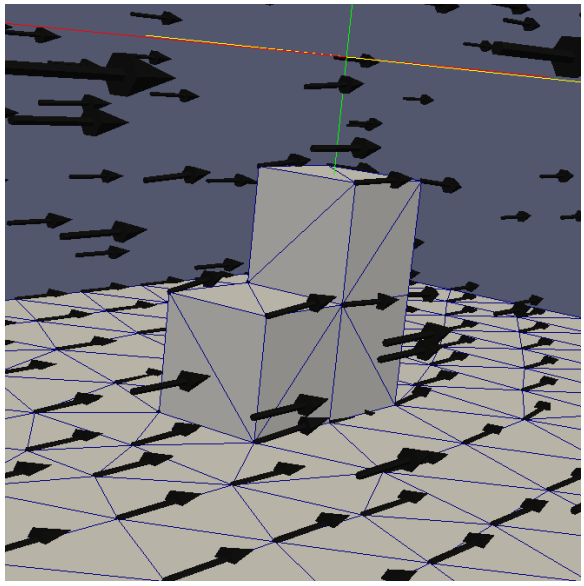


Figure 5.2: Results from the potential flow solver on the cubi model

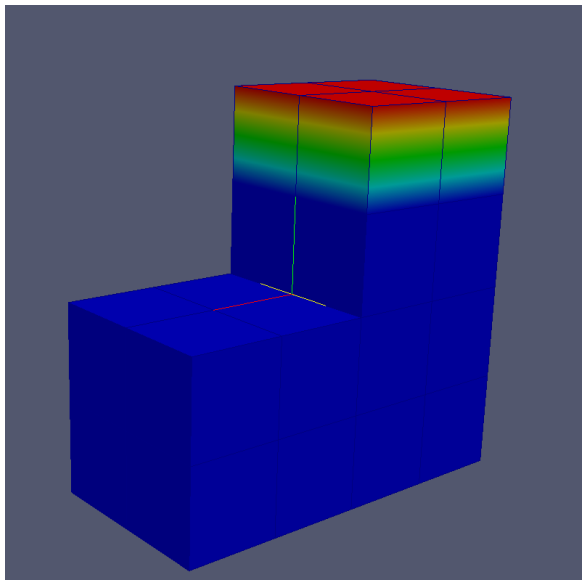


Figure 5.3: Final results from the thermal module solution of the cubi model at t=0.0 sec

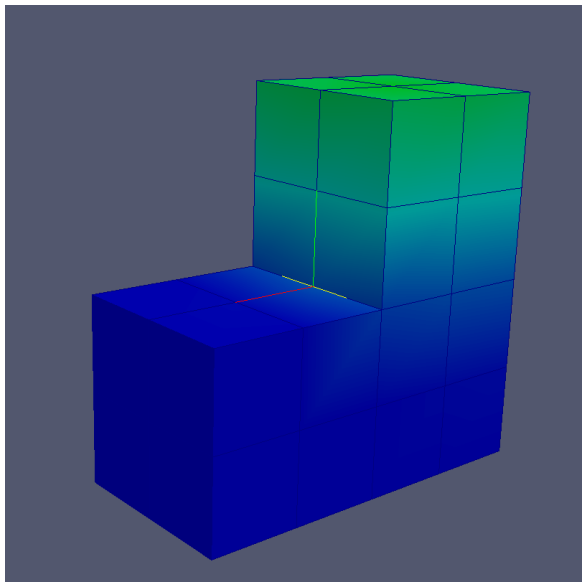


Figure 5.4: Final results from the thermal module solution of the cubi model at t=30.0 sec

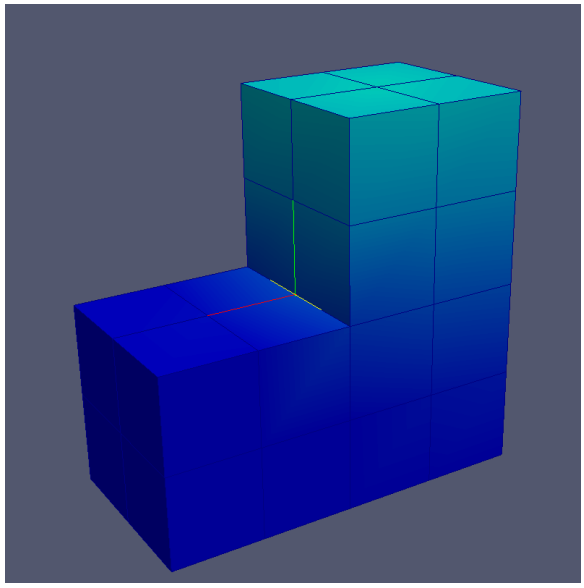


Figure 5.5: Final results from the thermal module solution of the cubi model at t=60.0 sec

Chapter 6

BioMechanical Module

The electro-mechanical module is a fully coupled finite element model of the heart that brings into account the effects of excitation due to electrical signal propagation and the effects caused by the ensuing deformation. This is a fully implicit scheme that, like the fluid module, uses the finite element method for the computational implementation

The cardiac system of the heart is a very complex system and in the past there have been several models developed to describe the electrical model. One of the most well known models for the electrical activity of the heart is the Beeler-Reuter model. The Beeler-Reuter model is used as the electrical model in the biomechanical module as well as the reaction-diffusion module in this study.

6.1 Governing Equations

The biomechanical module is governed by the reaction-diffusion equation and the stress equations given in equations 6.2 and 6.1 respectively. These equations describe the interaction of the spatial coordinates $\mathbf{x}$ and the transmembrane potential V. The variables are the conduction tensor D, the Kirchoff stress tensor σ, ion channels I_{ion}, the transmembrane capacitance C_m and the body force $\mathbf{b}$.

$$\nabla \cdot \sigma + b = 0 \tag{6.1}$$

$$\frac{\partial V}{\partial t} - \nabla \cdot (D \cdot \nabla v) + \frac{I_{ion}}{C} = 0 \tag{6.2}$$

Both equations contain natural boundary conditions for the surface domain of the heart. The reaction-diffusion equation (Eq.6.2) has the no-flux boundary condition in Eq. 6.3 imposed on the surface domain.

$$\frac{\partial V}{\partial n} = \nabla V \cdot n = 0 \quad on\, \partial\Omega \tag{6.3}$$

For the mechanical portion the natural boundary condition is

$$\sigma \cdot n = 0 \tag{6.4}$$

A third boundary condition is imposed on a portion of the surface domain, $\partial\Lambda$. This forced boundary condition is required for the problem to be well defined. In the case of this problem the boundary condition is placed on the portion of the tissue that must remain stationary, such as a point where the heart tissue would be attached to connective tissue.

$$x = \overline{x} \quad on \quad \partial\Lambda \tag{6.5}$$

Splitting Method

Depending on the electrical model used in the simulation, as many as 50 separate ODE's can be used to describe Eq. 6.2. To simplify the solution the splitting operator method is employed. This method involves breaking the reaction-diffusion equation into two parts by separating the ion channels thusly

$$I_{ion} = -(I_e + I_{sac}) \tag{6.6}$$

We can further denote the channels as two forcing functions

$$F_{ev} = \frac{I_e}{C_m} \quad , \quad F_{mv} = \frac{I_{sac}}{C_m} \tag{6.7}$$

Inserting Eq. 6.7 into Eq. 6.2 creates a modified reaction-diffusion equation.

$$\frac{\partial V}{\partial t} - \nabla_x \cdot (D \cdot \nabla_x V) - F_{ev} - F_{mv} = 0 \tag{6.8}$$

Continuing the splitting method Eq. 6.8 can then be broken into a PDE system and a separate ODE system for the electrical impulses.

$$\frac{dv}{dt} = F_{ev} \tag{6.9}$$

$$\frac{\partial V}{\partial t} - \nabla_x \cdot (D \cdot \nabla_x V) - F_{mv} = 0 \tag{6.10}$$

Eqns. 6.9 and 6.10 can then be solved in two independent steps. The first step assumes the initial transmebrane potential at time n is given as V_n. This potential value is then updated using Eq. 6.9 at time $n + 1/2$. This updated potential value is used as the initial condition for the solution of Eq. 6.10 using an implicit finite element scheme.

ODE Model

The ODE electrical model is the primary driving force for the stimulation of the electrical impulse in the tissue. The model takes on the form of an ODE Eq. 6.11.

$$\frac{dV}{dt} = -\frac{I_e}{C_m} \tag{6.11}$$

The current I_e is the total current traveling through the membrane and depending on the model, It can have as few as three channels or as many as 14 channels. A basic model is the Hodgkin-Huxley model, Eq. 6.12, and takes into account the Potassium, Sodium channels and the initial stimulation.

$$\frac{dV}{dt} = -\frac{I_{K_1} + I_{Na} + I_{stim}}{C_m} \tag{6.12}$$

A more advanced model was developed to take into account more of the ion channels present, forming the Beeler-Reuter model. In Eq. 6.13 is the governing ODE for the Beeler-Reuter Model. This model takes into account the calcium channel and a secondary potassium channel, x.

$$\frac{dV}{dt} = -\frac{I_{K_1} + I_{x_1} + I_{Na} + I_{Ca} - i_{external}}{C_m} \tag{6.13}$$

The solution of the ODE on a single cell is shown in Fig 6.1, which shows the Ion current values, gate values, calcium level and the resultant change in the action potential. The complete solution for the Beeler-Reuter model is covered in complete detail in Appendix D.

6.2 Finite Element Implementation

The conversion of the governing equations into a finite element scheme was originally performed by Xia Henian at the University of Tennessee [25].

6.2.1 Mechanical Model

The formulation of the mechanical model into the finite elements begins by multiplying Eq. 6.1 by a trial space function Ψ and integrating over the domain.

$$G_x = \int_\Omega \Psi \cdot \nabla_x \cdot \sigma dV + \int_\Omega \Psi \cdot b dV = 0 \tag{6.14}$$

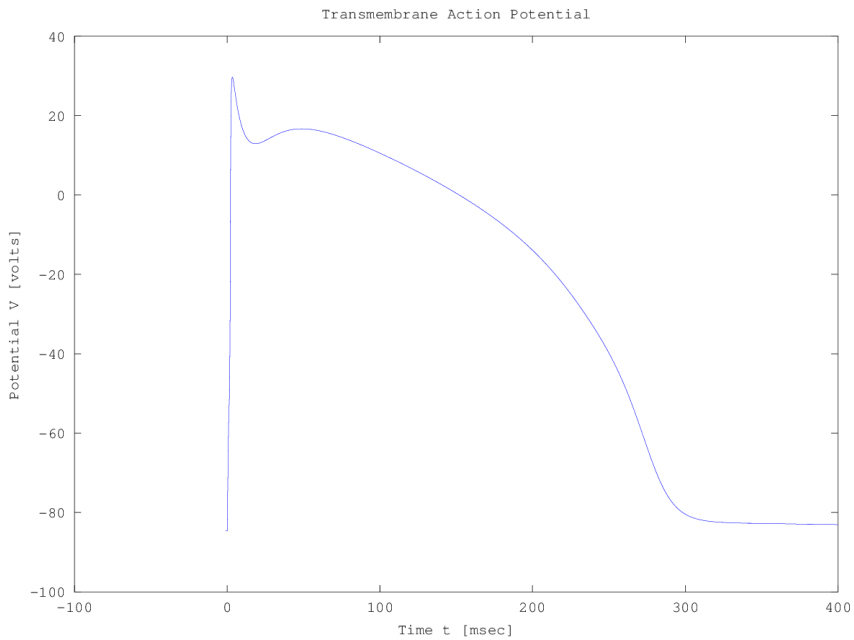


Figure 6.1: Results from Beeler-Reuter model run in a standalone MATLAB program

and then performing integration by parts

$$G_x = \int_\Omega \nabla_x(\Psi) : \sigma dV - \int_\Omega \Psi \cdot \sigma \cdot n da - \int_\Omega \Psi \cdot b dV = 0 \tag{6.15}$$

By applying the natural boundary condition a portion of 6.15 can then be eliminated.

$$G_x = \int_\Omega \nabla_x(\Psi) : \sigma dV - \int_\Omega \Psi \cdot b dV = 0 \tag{6.16}$$

Linearization of the system is conducted by determining the incremental change in G_x instead of generating a Jacobian matrix as done with the previous two modules.

$$G_x\left(x_{n+1}, V_{n+1}\right) = G_x + \Delta G_x\left(x_n, v_n; x_{n+1} - x_n, V_{n+1} - V_n\right) \tag{6.17}$$

Because the body force per volume is constant it can assumes the derivative of this

integral will be zero.

$$\Delta \int_{\Omega} \Psi \cdot b dV = 0 \tag{6.18}$$

The resultant change in G_x is then

$$\begin{aligned} \Delta G_x = \Delta \int_{\Omega} \nabla (\Psi) : \sigma dV = \int_{\Omega} \nabla (\Psi) : M : \nabla(\Delta x) dV \\ + \int_{\Omega} \nabla (\Psi) : [\nabla(\Delta x)\sigma] \, dV + \int_{\Omega} \nabla (\Psi) : Q \delta v dV \end{aligned} \tag{6.19}$$

In Eq. 6.19 M is the fourth-order tangent moduli of the stresses, Q is a voltage dependent function.

6.2.2 Electrophysiology Model

Formulation of the electrical model follows the same steps as the mechanical model. It begins by generating the Galerkin weak statement.

$$G_v = \int_{\Omega} \left[\Psi \frac{\partial V}{\partial t} + \nabla (\Psi) \cdot (D \cdot \nabla V) \right] dV - \int_{\partial\Omega} \Psi D \cdot \nabla V \cdot n da - \int_{\Omega} \Psi F_{mv} dV \tag{6.20}$$

The Neumann boundary conditions on the surface domains are zero and therefore discarded.

$$\int_{\Omega} \Psi D \cdot \nabla v \cdot n da = 0 \tag{6.21}$$

Linearizing as before, but this time only the part regarding the stretch activated channels requires linearization. The rest of the terms are linear.

$$G_v^1 = \int_{\Omega} \left[\Psi \frac{1}{2} + \nabla(\Psi) \cdot (D \cdot \nabla V) \right] dV \tag{6.22}$$

$$G_v^2 = -\int_\Omega \Psi F_{mv} dV \tag{6.23}$$

Therefore G_v^2 is

$$G_v^2\left(x_{n+1}, V_{v+1}\right) = G_v^2\left(x_n, v_v\right) - \int_\Omega \Psi\left(E : \left(\nabla(\Delta x)\right) + F\Delta V\right) dV \tag{6.24}$$

E and F are second-order tensors.

6.2.3 Computational Implementation

The preceding set of equations are then converted into a format used to fill the linear system of equations in the Ax=b form. In the following equations, the number of nodes is N and the number of nodes per element is n_{en}. The global index is represented by the value g, m and n is the number of dimensions, and l is the element id. These equations are assembled on an element by element basis and then inserted into the compressed-row Epetra matrices.

$$\begin{aligned}
A\left(3\left(g_I^l\right)+m, 3\left(g_J^l-1\right)+n\right) &+= \sum_{K=1}^{n_{en}}\sum_{n=1}^{n_d}\int_{\Omega_e^h} \nabla N^I : N^K\left[M_{m,n}^K\right] : \nabla N^J dV \\
A\left(3\left(g_I^l\right)+m, 3\left(g_J^l-1\right)+m\right) &+= \sum_{K=1}^{n_{en}}\int_{\Omega_e^h} \nabla N^I \cdot \left(\nabla N^J \cdot N^K \sigma^K\right) dV \\
A\left(3\left(g_I^l\right)+m, 3N+g_J^l+m\right) &+= \sum_{K=1}^{n_{en}}\sum_{n=1}^{n_d}\int_{\Omega_e^h} \nabla N^I \cdot N^K Q_m^K N^J dV \\
Rhs\left(3\left(g_I^l\right)+m\right) &+= \sum_{K=1}^{n_{en}}\int_{\Omega_e^h} \nabla N^I N^J \cdot N^K \sigma_m^J dV
\end{aligned} \tag{6.25}$$

$$
\begin{aligned}
&A\left(3N+g_I^l, 3\left(g_J^l\right)+m\right)+=-\Delta t\sum_{J=1}^{n_{en}}\sum_{K=1}^{n_d}\int_{\Omega_e^h} N^I N^K E_m^K\cdot\nabla N^J dV \\
&A\left(3N+g_I^l, 3N+g_J^l\right)+=\int_{\Omega_e^h}\left(N^I N^J-\Delta t\sum_{K=1}^{n_{en}} N^I N^K F^K N^J\right)dV \\
&+\Delta t\theta\sum_{K=1}^{n_{en}}\int_{\Omega_e^h}\nabla N^I\cdot\left(N^K D^K\cdot\nabla N^J\right)dV \qquad (6.26)\\
&Rhs\left(3N+g_I^l\right)+=-\Delta t\sum_{J=1}^{n_{en}}\sum_{K=1}^{n_{en}}\int_{\Omega_e^h}\nabla N^I\cdot\left(N^K D^K\cdot\nabla N^J\right)dV v^J \\
&+\Delta\int_{\Omega_e^h}\sum_{J=1}^{n_{en}} N^I N^J I_{sac}^J dV
\end{aligned}
$$

6.3 Module Structure

The biomechanical module has been written using C++ and implements and object-oriented method for the data structures and execution of the solvers. It also makes use of the Trilinos library for the linear algebra data structures in Epetra and the sparse matrix solver AztecOO. Figure 6.2 illustrates the workflow of the module during runtime.

While the fluid and thermal modules use of the PT for domain decomposition, the biomechanical module has its own tools to partition the mesh. The partitioner uses METIS for the domain decomposition. Each process executes the partitioner and reads in its respective local data.

The module begins with the process of initializing all the variables and data structures. The boundary conditions are defined based on a hard coded location for the area of stimulation and the electrical models are initialized depending on the model used. The deformation gradient and conduction tensors are initialized followed by the pacing protocols.

The time integration begins by solving for the updated potential value from the ODE models. Then the assembly process begins over each element of the system. The matrix data is stored in compressed-row storage in an Epetra matrix. Once the boundary conditions are

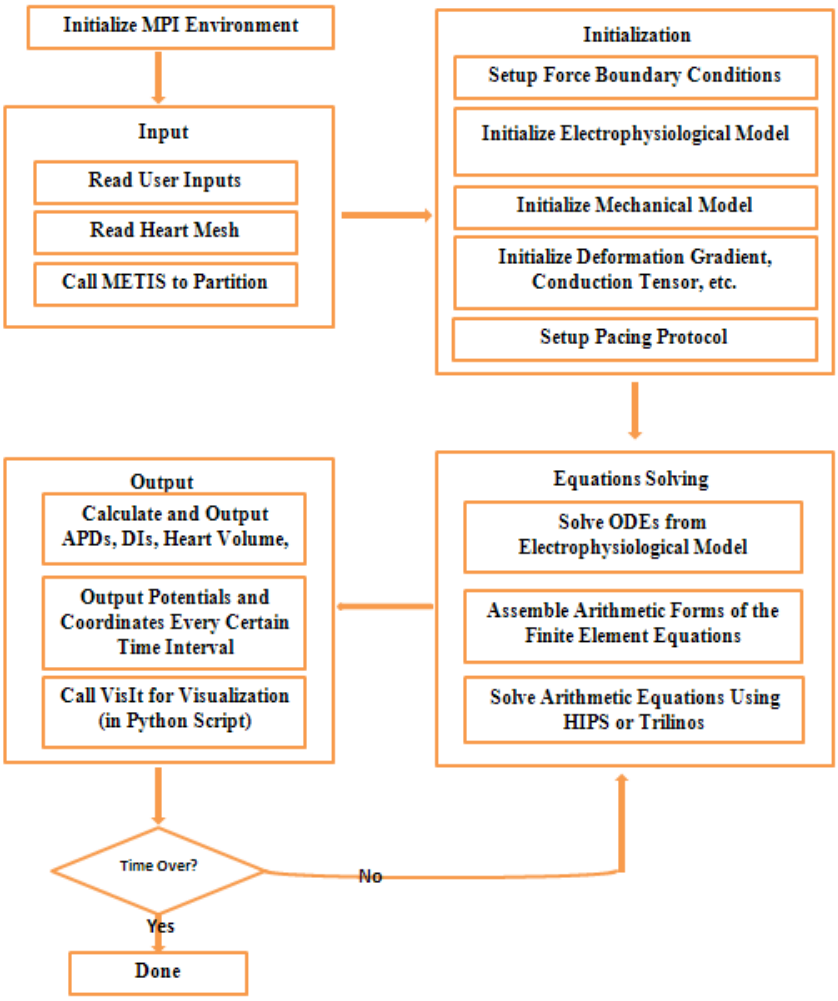


Figure 6.2: Structure and flow of the biomechanical module

applied AztecOO is called.

The spatial coordinates and action-potential are then updated and the time step incremented. After each timestep the data is exported into a tecplot file for visualization. Also at the end of each timestep a restart file is generated which can be used to start the system from a previous timestep in the simulation.

6.4 Mesh Module

The mesh module is designed to parse an input mesh file containing the connectivity of the surfaces of the fluid domain and update the nodes from a tecplot file. With the update nodal coordinates it generates a new internal tetrahedral mesh. This mesh is then exported to a Patran file that can be parsed by the PT.

The module requires the input mesh for the system containing the three dimensional mesh for the tissue and a subset of surfaces for the fluid domain. As the input file is read the nodal ID's are saved as a map to be reused at each call of the module.

At the first call of the module, the Gmsh library is used to generate a configuration file containing the geometrical and boundary data. At each subsequent time step the coordinate data saved in this file is updated. This file can then be directly read by Gmsh and used to generate a new internal mesh. Afterwards the new mesh and boundary conditions are exported to the Patran file.

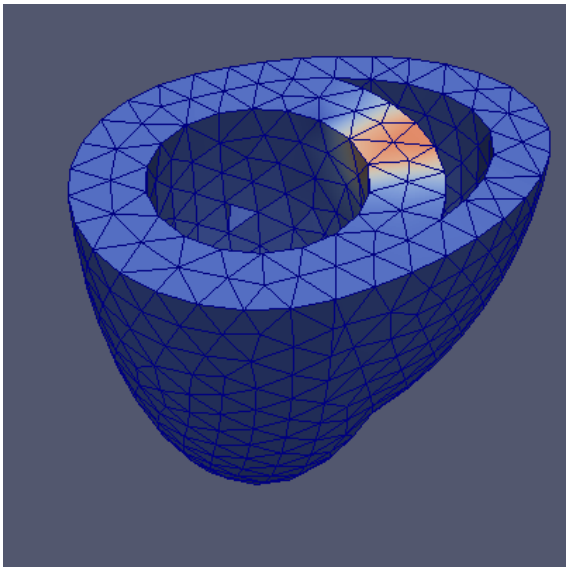
6.5 Reaction-Diffusion Module

A new module has been developed from the reaction-diffusion equation set. This module is designed to handle only the change in action-potential in tissue an will ignore the mechanical portion of the biomechanical module.

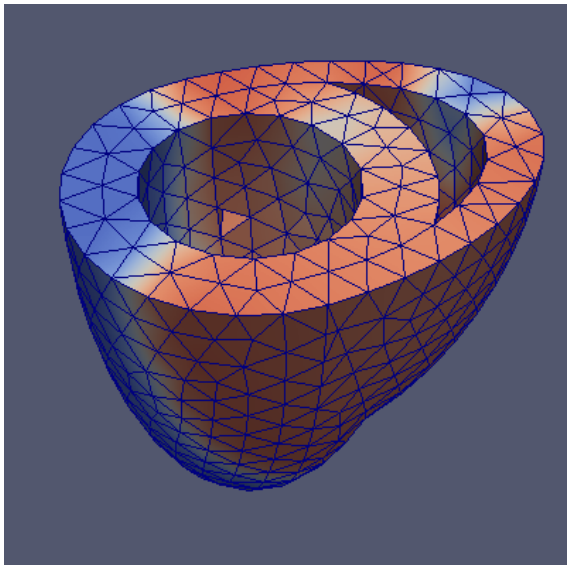
The reason for build a new module stems from the inherent difficulties in generating complex geometries using unstructured hexahedral mesh generators. With the implemen-

tation of a new module it is possible to conduct simulations using triangle, quadrilateral, tetrahedral and hexahedral unstructured grids.

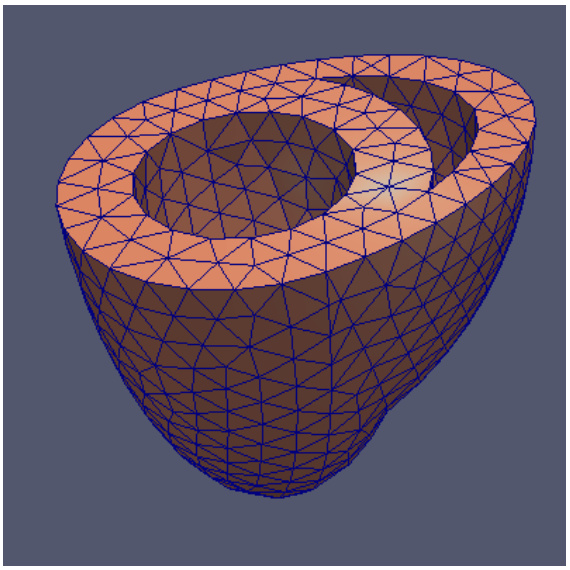
Another benefit of the new module is the ease is implementing new ODE modules. The module is written in C++ and makes use of certain object-oriented features. This allows one to generate new ODE models using the base class provided and the currently built models as guide. Currently only the Beeler-Reuter model is built.



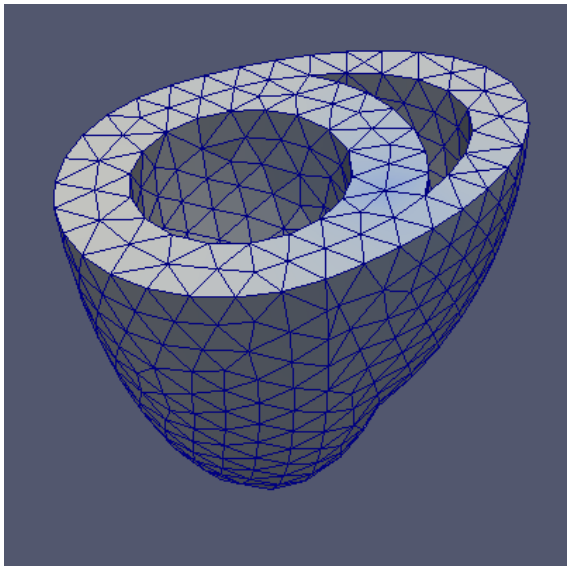
(a) Initial stimulation



(b) Results of the simulation after 80 ms.



(c) Results of the simulation after 160 ms.

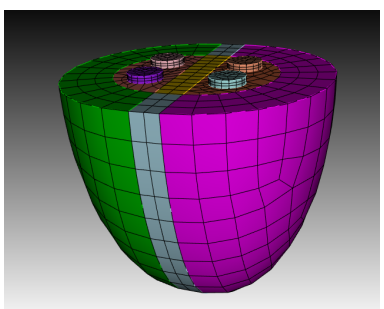


(d) Results of the simulation after 240 ms.

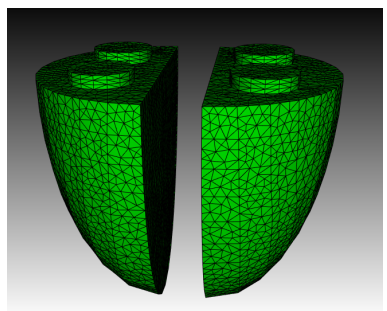
Figure 6.3: Simulation of a basic heart model using the Reaction-Diffusion Module

6.6 Test Case Results and Discussion

The heart model used for the biomechanical test case is shown in figure 6.4a. This mesh consists of hexahedron for the heart tissue domain, but for the fluid domain only surface elements are defined. Once the first timestep of the biomechanical module have completed, the updated nodal positions and connectivity are passed to the mesh module. The mesh module then generates a new internal tetrahedral mesh as illustrated in figure 6.4b.



(a) Initial heart tissue geometry

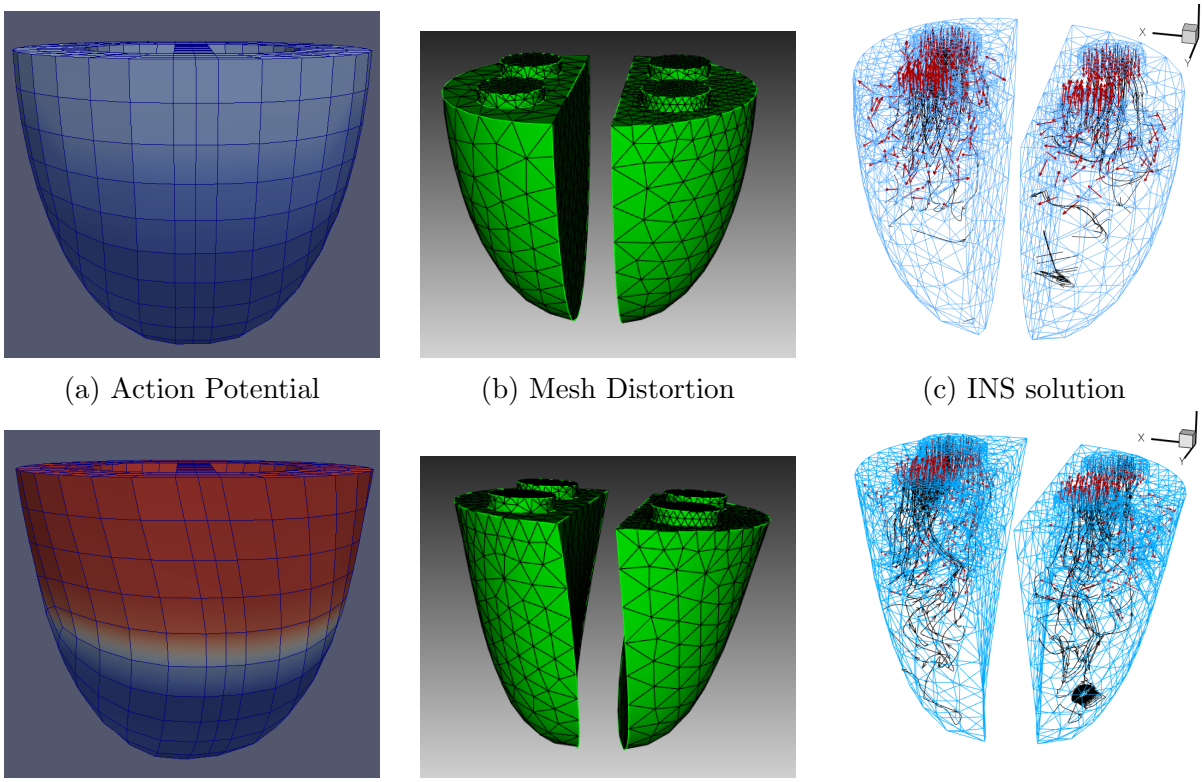


(b) Initial internal flow geometry

Figure 6.4: Original geometries used for the biomechanical test case

The following figures are the results for the biomechanical test case using the models in 6.4. The details for the driver and configuration files can be found in Appendix B.2

Figure 6.5 illustrates the results for the biomechanical test case. From left to right the is the results of the biomechanical module, then the newly generated mesh and then the results for the Incompressible Navier-Stokes solver.



(a) Action Potential (b) Mesh Distortion (c) INS solution

(d) Action Potential (e) Mesh Distortion (f) INS solution

Figure 6.5: Simulation at t = 5 ms

Looking at the figures above the action-potential can be clearly seen traversing the cardiac tissue in figures a and d. The resultant deformation is also readily evident as the entire lower body of the model twists to the right. To the right in figures b and e the update internal mesh is shown. Here the mesh module has taken the results from the biomechanical module and updated only the nodes that have deformed. It then uses the Gmsh library to regenerate an internal tetrahedral mesh. This mesh is then converted to the input file for the fluid module. In figures c and f the deformation and fluid flow characteristics are clearly visualized.

The cardiac simulations have also been tested on models more closely resembling a heart. One such test is shown in figure 6.6. In this simulation we are only able to demonstrate the

working of the biomechanical module due to difficulties with the original mesh and Cubit. Looking closely at top of figure 6.6 the module is run with the mechanical stresses deactivated. This shows the propagation of the action potential similar to the heart model in figure 6.3. With the mechanical component turned on the resultant deformation is shown in the lower portion of figure 6.6. At the end of each beat the tissue relaxes and returns to its original shape.

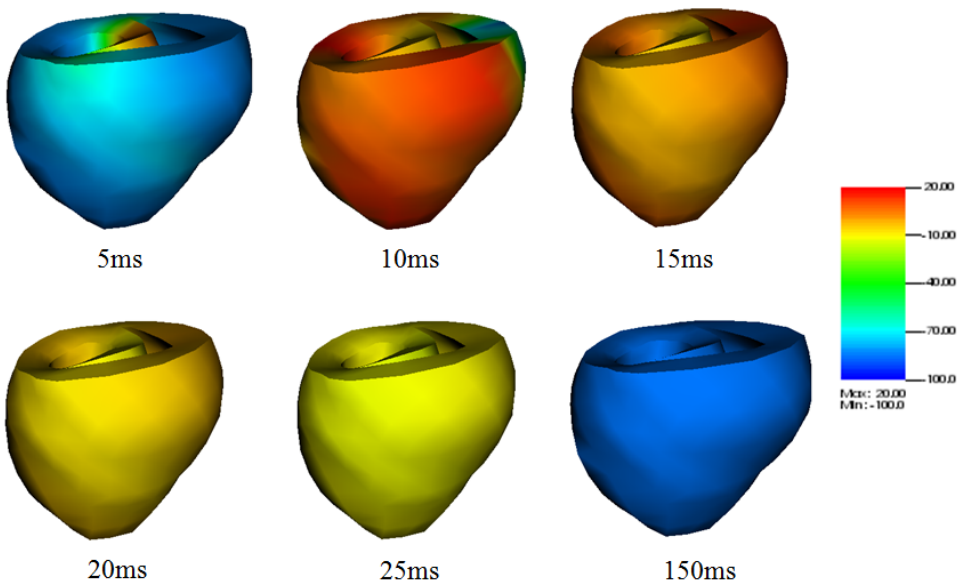


Figure 6.6: Simulation of cardiac tissue using realistic geometry

In this test case the capability of the IEL has been demonstrated to run multiple simulations, and to schedule each individual simulation to solve a complex physics problem. This has also demonstrated how it is possible to integrate third party software into the IEL and use it as a physics module. Future work intends to develop and add new ODE models to the standalone ode module and increase the coupling between the biomechanical and fluid modules.

Chapter 7

Conclusion

In this thesis I have expanded the capability of the Interoperable Executive Library by integrating new physics and utility modules. Three of the modules are derived from the same finite element solver designed to run on large scale parallel computers. The new modules include a potential flow solver, an Incompressible Navier-Stokes solver, and a reaction-diffusion equation solver.

An external biomechanical solver used to simulate the electrophysiological response of cardiac tissue has been integrated to work with the IEL. In conjunction with this module a mesh module was developed to connect the biomechanical module with the fluid flow module.

These modules were then combined into two multiphysics test cases. A thermal-fluid test case utilizes a thermal module combined with the potential flow and radiosity module. In this test case it was demonstrated how the IEL could be used to handle communication of shared boundary data between multiple modules running concurrently.

The second test case couples the biomechanical module, the pre-processing tool, the mesh module and the Incompressible Navier-Stokes solver. This test case demonstrated advanced scheduling techniques using the driver program. It also showcased the ability to integrate third-party software into the IEL with minimal effort.

The work conducted in this thesis has expanded the scope of applications the IEL can

simulate. It has also demonstrated the basic capabilities of the IEL to schedule, manage communication, and execute large scale multiphysics simulations.

Bibliography

[1] Xiangmin Jiao, Gengbin Zheng, Phillip A. Alexander, Michael T. Campbell, Orion S. Lawlor, John Norris, Andreas Haselbacher, and Michael T. Heath. A system integration framework for coupled multiphysics simulations. Technical report, 2006.

[2] COMSOL. www.comsol.com, 2013.

[3] Dune. www.dune-project.org, 2013.

[4] OpenFOAM Foundation. *OpenFOAM The Open Source CFD Toolbox*, 2.1.1 edition, 5 2012.

[5] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, and John Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[6] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006.

[7] Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. *FFC: the FEniCS Form Compiler*, chapter 11. Springer, 2012.

[8] Sebastian Krittian, Uwe Janoske, Herbert Oertel, and Thomas Bohlke. Partitioned fluid-solid coupling for cardiovascular blood flow. Technical report, Institute for Fluid Mechanics, Karsruhe Institute of Technology, 2010.

[9] Michael A. Heroux and James M Willenbring. *Trilinos User Guide*. Sandia National Laboratories, Albuquerque, NM 87185-1110, September 2007.

[10] Jack Dongarra. *MAGMA User's Guide*. Innovative Computing Laboratory, November 2009.

[11] Scalapack. http://netlib.org/scalapack/slug/index.html.

[12] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2013. http://www.mcs.anl.gov/petsc.

[13] Sandia National Laboratories. *Cubit 13.2 User Documentation*.

[14] Paraview. Paraview. www.paraview.org, 2013.

[15] Christophe Geuzaine and Jean-Fran cois Remacle. *Gmsh Reference Manual*, July 2013.

[16] Tecplot. http://www.tecplot.com/.

[17] Visit visualization tool. https://wci.llnl.gov/codes/visit.

[18] Elton Lewis Freeman. Validation of weak form thermal analysis algorithms supporting thermal signature generation. Master's thesis, University of Tennessee, December 2012.

[19] George N. Walton. Calculation of obstructed view factors by adaptive integration. Technical report, National Institute of Standards and Technology, 2002.

[20] J. N. Reddy. *An Introduction to the Finite Element Method*. McGraw-Hill, 1993.

[21] Alan J. Baker. *The Computational Engineering Sciences*. j-Computek Press, 2006.

[22] Frank M. White. *Viscous Fluid Flow*. McGraw-Hill International, third edition, 2006.

[23] Paul T. Williams. *The Continuity Constraint Method*. PhD thesis, University of Tennessee, 1993.

[24] Frank P. Incropera and David P. DeWitt. *Introduction to Heat Transfer*. Wiley, 2002.

[25] Xia Henian, Kwai Wong, and Xiapeng Zhao. Parallel fem simulation of electromechanics in the heart. Technical report, University of Tennessee, 2011.

Appendix

Appendix A

IEL Definitions

A.1 Data Structures

The IEL contains several data structurse to store information about the simulation, data handles and MPI communicators. The primary data structure is *iel_component_info_t* and is composed of the following data fields.

- MPI_Comm comm
- int server_rank
- int my_rank
- int size
- provides [IEL_MAX_DEPEND]
- requires [IEL_MAX_DEPEND]
- pcount [IEL_MAX_DEPEND]
- int nump
- int numr

- star_data_handle_t* phandle[STAR_MAX_DEPEND]
- star_data_flow_t* data_flow

A.1.1 Detailed Description

MPI_Comm comm

MPI communicator for this module

int size

The total number of ranks in the comm system

int server_rank

Server rank in the communicator 'comm'

int my_rank

The component's MPI rank

int provides[]

The data handle ID's required in the module

int requires[]

The data handle ID's provided in the module

int pcount[]

The number of receivers for each data handle

int nump

The number of entries in the provides array

int numr

The number of entries in the requires array

iel_data_handle_t * phandle[]

Pointers to sent handles

iel_data_flow_t * data_flow[]

Pointer to the global data flow

Another One

The data type *iel_data_flow_t* is in itself a data structure describing the data handles used for information exchange. The handle acts as a connector between two modules and is accessed using functions defined in the proceding section. **int id**

The handle's ID

size_t size

The size (in bytes) of the handle's buffer

int num_bufs

Currently Unused

void * buf

Pointer to copy of the module's data

int dest[]

List of destinations for this handle

int complete[]

List of completion status indicators

MPI_Request mpi_req[]

List of MPI_Requests

A.2 Communicator Library

int iel_get (iel_component_info_t * *component_info*, iel_data_handle_t * *handle*, void * *data*)

Read data from the specified data handle. Initiates communication with the providing module.

int iel_get_many (iel_component_info_t * *component_info*, int *num_handles*, iel_data_handle_t * *handle*, void ** *data*)

Retrieves data from multiple data handles. Can be more efficient than multiple calls to iel_get by avoiding unnecessary blocking calls.

int iel_wait_prev_puts (iel_component_info_t * *component_info*, iel_data_handle_t * *handle*)

Waits for all previous calls to iel_put to complete before continuing.

int star_finalize (iel_component_info_t * *component_info*)

Called by all running modules to ensure proper exit.

int iel_put (iel_component_info_t * *component_info*, iel_data_handle_t * *handle*, void * *data*)

Write data to the specified data handle. Initiates communication with the providing module.

int ile_comm_create (MPI_Comm *comm*, int * *ranks*, int *new_size*, MPI_Comm * *new_comm*)

Creates a new MPI communicator for use in generating subset ranks. If this function is called, then all ranks under 'comm' must also call it.

int iel_init_component (star_component_info_t * *component_info*, int *my_rank*, int *server_rank*, int *size*, MPI_Comm *comm*)

Initializes the component_info structure with information about this node

int iel_provide (iel_component_info_t * *component_info*, iel_data_handle_t * *handle*)

Declares that this module and process provides the specified data handle.

int iel_require (iel_component_info_t * *component_info*, iel_data_handle_t * *handle*)

Declares that this module and process requires the specified data handle.

int iel_write_dot_file (char * *fname*, iel_component_info_t * *component_info*)

Writes the global data flow to a 'dot' file.

int iel_data_flow_init (iel_data_flow_t * *data_flow*, size_t *size*)

Initializes the global data flow structure.

int iel_gather_requirements (iel_component_info_t * *component_info*)

Gathers the dependency information from all the processes and assembles the global data flow.

int iel_component_destruct (iel_component_info_t * *component_info*)

Frees the resources associated with the provided component.

iel_data_handle_t *iel_new_data_handle(int *id*,size_t *size*,int *num_bufs*)

Allocates memory for a new data handle.

iel_data_handle_destruct(iel_data_handle_t **handle*)

Free's resources associated with the specified handle. Does not free the handle itself.

Appendix B

Test Case Drivers and Configuration Files

B.1 Thermal-Fluid Test Case

```
#include <stdlib.h>
#include <stdio.h>
#include "iel_exec_info.h"
#include "iel-pt.h"

void CreateConfigFile()
{
  FILE *fin;
  fin = fopen("ielpt.cfg","w");
  fprintf(fin,"shared_bc_sizes = [1];\n\n");
  fprintf(fin,"modules=(\n\t{\n");
  fprintf(fin,"\t\tfunction=\"ielPT\";\n");
  fprintf(fin,"\t\tlibtype=\"static\";\n");
  fprintf(fin,"\t\targs=();\n");
  fprintf(fin,"\t\tsize=1;\n");
  fprintf(fin,"\t\tpoints=(\n\t\t\t(())\n\t\t);\n\t}\n);\n");
  fclose(fin);
}

int main(int argc, char*argv[])
{
  int rc,rank;
```

```
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  ielAddModule(&ielPT,"ielPT");

  if(rank==0){
    fprintf(stderr,"Creating partitioner config file\n");
    CreateConfigFile();
    fprintf(stderr,"Calling partitioner...\n");
  }
  MPI_Barrier(MPI_COMM_WORLD);

  rc = ielExecutive(MPI_COMM_WORLD, "ielpt.cfg");
  if(rank==0) fprintf(stderr,"Completed iel-pt..waiting\n");

  MPI_Barrier(MPI_COMM_WORLD);

  if(rank==0) fprintf(stderr,"Beginning main program...\n");
  rc = ielExecutive(MPI_COMM_WORLD, "iel-exec.cfg");
  if(rank==0) fprintf(stderr,"Completed.\n");

  MPI_Finalize();
  return rc;

}


shared_bc_sizes = [224,224,224,224];

modules = (
  {
    function="thermal_module";
    library="static";
    args=();
    size=2;
    points=(
      (([0,224]),()),
      (() ,([0,224])),
      (([0,224]),()),
      (() ,([0,224]))
    );
  },
  {
    function="Radiosity_Module";
    library="static";
    args=("view3D.indat","cubi2inv.dat");
    size=2;
    points=(
      (([0,224]),()),
```

```
      (() ,([0,224])),
      (() ,())),
      (() ,())
    );
  }
  {
    function="potential_odule";
    library="static";
    args=();
    size=2;
    points=(
      (() ,())),
      (() ,())),
      (([0,224]) ,())),
      (() ,([0,224]))
    );
  }
);
```

B.2 Biomechanical Test Case

```
shared_bc_sizes = [1];

modules=(
  {
    function="EPModule";
    args=("filler","config_file");
    libtype="static";
    size=2;
    points=(
        (())),
        (())
    );
  }
);
```

```
shared_bc_sizes = [1];

modules=(
  {
    function="MeshUpdate";
    args=("filler","config_file");
    libtype="static";
    size=1;
    points=(
        (())
```

```
    );
  }
);


shared_bc_sizes = [1];

modules=(
  {
    function="StarPT";
    args=();
    libtype="static";
    size=1
    points=(
      (())
    );
  }
);


shared_bc_sizes = [1];

modules=(
  {
    function="INSModule";
    args=("filler","config_file");
    libtype="static";
    size=2;
    points=(
        (()),
        (())
    );
  }
);
```

Appendix C

Thermal Module Formulation

Converting the Galerkin Weak Statement and applying all of the boundary conditions gives

$$F_T = [m200]\left(\{T\}^{n+1} - \{T\}^{n}\right) + \frac{\Delta t}{2}\left[\begin{array}{c} k\,[m2kk]\left(\{T\}^{n+1} + \{T\}\right) \\ + \left(\frac{1}{\rho C_p}\right)\{h_{conv}\}^T\,[n3000]\left(\{T\}^{n+1} + \{T\}^{n}\right) \\ - \left(\frac{1}{\rho C_p}\right)\{h_{conv}\}^T\,[n3000]\left(\{T_{amb}\}^{n+1} + \{T_{amb}\}^{n}\right) \\ + \left(\frac{\sigma}{\rho C_p}\frac{\epsilon_e}{1-\epsilon_e}\right)[n200]\left(\{T\}^{4,n+1}\{T\}^{4,n}\right) \\ - \left(\frac{\sigma}{\rho C_p}\frac{\epsilon_e}{1-\epsilon_e}\right)[n200]\left(\{T_{sky}\}^{4,n+1} + \{T_{sky}\}^{4,n}\right) \\ + \left(\frac{\sigma}{\rho C_p}\right)\frac{1}{1-\epsilon_e}[n200]\left(\{T_{grnd}\}^{4,n+1} + \{T_{grnd}\}^{4,n}\right) \\ + \left(\frac{\sigma}{\rho C_p}\right)\frac{1}{1-\epsilon_e}[n200]\left(\{R\}^{4,n+1}\{R\}^{4,n}\right) \\ - [m200]\left(\{SRC\}^{n+1}\{SRC\}^{n}\right) \\ + \left(\frac{\epsilon}{\rho C_p}\right)[n200]\left(\{FLUX\}^{n+1}\{FLUX\}^{n}\right) \end{array}\right] \tag{C.1}$$

And the Jacobian being

$$\frac{\partial F_T}{\partial T} = [m200] + \frac{\Delta t}{2} \begin{bmatrix} k\,[m2kk] \\ + \left(\frac{1}{\rho C_p}\right) \{h_{conv}\}^T\,[n3000]\left(\{T\}^{n+1}\right) \\ + \left(\frac{4\sigma}{\rho C_p}\frac{\epsilon_e}{1-\epsilon_e}\right)[n200]\left(\{T\}^{3,n+1}\right) \\ - \left(\frac{4\sigma}{\rho C_p}\frac{\epsilon_e}{1-\epsilon_e}\right)[n200]\left(\{T_{sky}\}^{3,n+1}+\right) \\ + \left(\frac{4\sigma}{\rho C_p}\right)\frac{1}{1-\epsilon_e}[n200]\left(\{T_{grnd}\}^{3,n+1}+\right) \end{bmatrix} \tag{C.2}$$

Appendix D

ODE Models

The driving force behind the electrical simulation of the heart are the ODE solution for the Ion channels. The models presented here are derived from the work performed by Hodgkin-Huxley which pioneered our understanding of the electrical activity of the heart.

D.1 Beeler-Reuter Model

The Beeler-Reuter Model was developed in 1977 to further the development of accurate models of the transmebrane action potential of the mammalian heart. The model revolves around a series of simple ODE's to calculate the change in voltage. Equation D.1 gives the fundamental equation for the BR-Model.

$$\frac{dV}{dt} = -\frac{I_{K_1} + I_{x_1} + I_{Na} + I_{Ca} - I_{external}}{C_m} \tag{D.1}$$

$$C_m = 1 \tag{D.2}$$

Each of the Ion channel in Equation D.1 are represented by either a polynomial or ordinary differential equation. Each channel may also contain a series of activation gates represented by a common ODE and individual $\alpha \quad \beta$ values. They are all presented below.

Currents

$$I_{K_1} = 0.35\{\frac{4\{exp[0.04(V+85)]-1\}}{exp[0.08(V+53)]+exp[0.04(V+53)]} + \frac{0.2(V+23)}{1-exp[-0.04(V+23)]}\} \tag{D.3}$$

$$I_{X_1} = x_1 \cdot 0.8\{\frac{exp[0.04(V+77)]-1}{exp[0.06(V+35)]}\} \tag{D.4}$$

$$I_{Na} = \left(\overline{g_{Na}} m^3 hj + g_{NaCa}\right)(V - E_{Na}) \tag{D.5}$$

$$I_{Ca} = \overline{g_{Ca}} df\,(V - E_{Ca}) \tag{D.6}$$

Gate values

$$\frac{dx_1}{dt} = \alpha_{x_1}(1-x_1) - \beta_{x_1} x_1 \tag{D.7}$$

$$\frac{dm_1}{dt} = \alpha_{m_1}(1-m_1) - \beta_{m_1} m_1 \tag{D.8}$$

$$\frac{dh_1}{dt} = \alpha_{h_1}(1-h_1) - \beta_{h_1} h_1 \tag{D.9}$$

$$\frac{dj_1}{dt} = \alpha_{j_1}(1-j_1) - \beta_{j_1} j_1 \tag{D.10}$$

$$\frac{d_1}{dt} = \alpha_{d_1}(1-h_1) - \beta_{d_1} d_1 \tag{D.11}$$

$$\frac{df_1}{dt} = \alpha_{f_1}(1-f_1) - \beta_{f_1} f_1 \tag{D.12}$$

alpha beta values

Table D.1: Table: Rate constants for α and β

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$(msec^{-1})$	$(msec^{-1})$	(mV^{-1})	(mV)	$((mV \cdot msec)^{-1})$	(mV)	(mV^{-1})	
α_{x_1}	0.0005	0.083	50	0	0	0.057	1
β_{x_1}	0.0013	-0.06	20	0	0	-0.04	1
α_m	0	0	47	-1	47	-0.1	-1
β_m	40	-0.056	72	0	0	0	0
α_h	0.126	-0.25	77	0	0	0	0
β_h	1.7	0	22.5	0	0	-0.082	1
α_j	0.055	-0.25	78	0	0	-0.2	1
β_j	0.3	0	32	0	0	-0.1	1
α_d	0.095	-0.01	-5	0	0	-0.072	1
β_d	0.07	-0.017	44	0	0	0.05	1
α_f	0.012	-0.008	28	0	0	0.15	1
β_f	0.0065	-0.02	30	0	0	-0.2	1

$$\alpha(\beta) = (C_1 exp[C_2(V + C_3)] + C_4(V + C_5))/(exp[C_6(V + C_3)] + C_7) \tag{D.13}$$

table of coefficients

Other Values

$$\overline{g_{Na}} = 4, g_{NaCa} = 0.003, E_{Na} = 50 \tag{D.14}$$

$$\overline{g_{Ca}} = 0.09, E_{Ca} = -82.3 - 13.0287 ln[Ca^{2+}]_i \tag{D.15}$$

Calcium ODE

$$\frac{d\left[Ca^{2+}\right]}{dt} = -10^{-7} i_{Ca} + 0.07\left(10^{-7} - \left[Ca^{2+}\right]_i\right) \tag{D.16}$$

Solution is based on taylor series

$$V^{n+1} = V^n + \frac{dV}{dt}\delta t + O(\delta t^2) \tag{D.17}$$

The solution of the ODE's

$$y = \left(y_o - \frac{\alpha_y}{\alpha_y + \beta_y}\right) e^{-(\alpha_y+\beta_y)t} + \frac{\alpha_y}{\alpha_y + \beta_y} \tag{D.18}$$

Calcium solution

$$\frac{d\left[Ca^{2+}\right]}{dt} = -10^{-7}\overline{g_{Ca}}df\left(V + 82.3 + 13.0287ln[Ca^{2+}]_i\right) + 0.07\left(10^{-7} - \left[Ca^{2+}\right]_i\right) \tag{D.19}$$

initial conditions

$$\begin{aligned} x_1 &= 0.0074 \\ m &= 0.0011 \\ h &= 0.99869 \\ j &= 0.99887 \\ d &= 0.0001 \\ f &= 0.983 \\ [Ca^2] &= 0.0000001 \end{aligned} \tag{D.20}$$

The system is solved using a simpel Forward Euler approximation derived from the Taylor-Series expansion. An example solution of this case is presented in the matlab code below.

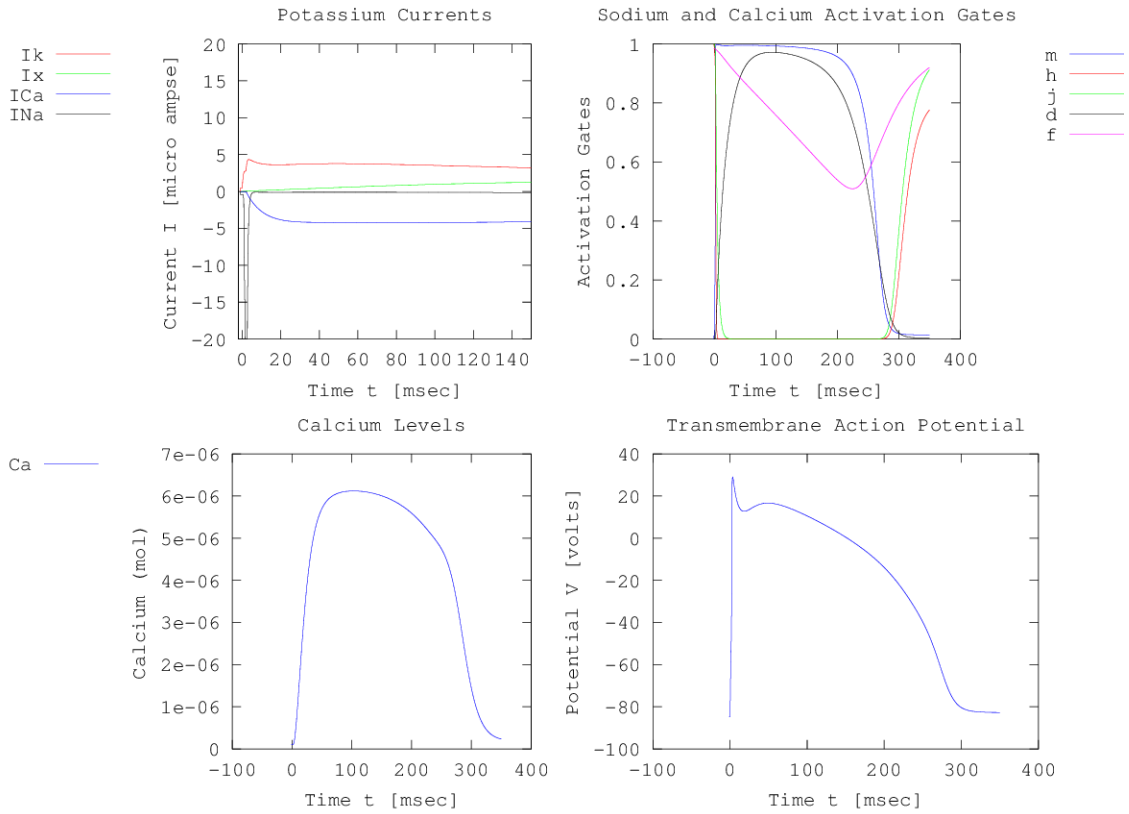


Figure D.1: Results from Matlab simulation of Beeler-Reuter Model

```
#!/usr/bin/octave -qf
clear all
clc
%addpath("/home/akail/AKDEV/ELECT-TISS/Ode-Test/");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matlab script for use in testing
% electrical model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize variables
Initial_V = -84.62; % Initial action potential value
Voltage = Initial_V;
Cm=1.0; % Membrane conductance
stimtime = 1 % Duration of the stimulation
```

```
% Time variables
ctime = -1.0; % current time set to the start point
runtime = 350.0; % total runtime
timestep = 0.02; % ode timestep

totalsteps = runtime/timestep;

counter = 2;
% Initialize arrays
voltagearray = zeros(1,totalsteps);
timearray = zeros(1,totalsteps);
Ixarray = zeros(1,totalsteps);
Ikarray = zeros(1,totalsteps);
voltagearray(1)=Voltage;
timearray(1)=ctime;

Gates = zeros(totalsteps,7);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set initial gate values
Gates(1,1)=0.0074; % X1 gate
Gates(1,2)=0.0011; % m gate
%Gates(1,2)=0.00024676; % m gate
Gates(1,3)=0.99869; % h gate
Gates(1,4)=0.99887; % j gate
Gates(1,5)=0.0001; % d gate
Gates(1,6)=0.983; % f gate
Gates(1,7)=0.0000001; % Calcium
%Gates(1,7)=0.0472; % Calcium
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Time loop
while ctime <= runtime

    % If ctime is during this period, the cell is stimulated
    if (ctime >= 0.0 && ctime <= stimtime)
        Jstim=30;
    else
        Jstim=0.0;
    end

    % Calculate mineral currents
    Ik =Kcurrent(Voltage);
    [Ix,Gates(counter,1)]=Xcurrent(Voltage,timestep,Gates(counter-1,1));

[INa,Gates(counter,2),Gates(counter,3),Gates(counter,4)]=Nacurrent(Voltage
    ,
timestep,Gates(counter-1,2),Gates(counter-1,3),Gates(counter-1,4));
```

```
[ICa,Gates(counter,5),Gates(counter,6),Gates(counter,7)]=Cacurrent(Voltage
    ,
timestep,Gates(counter-1,5),Gates(counter-1,6),Gates(counter-1,7));

    % Update the action potential
    Voltage = Voltage - timestep*(Ik+Ix+INa+ICa-Jstim)/Cm;

    %printf ('V = %f, Ik = %f, Ix = %f, INa = %f, ICa = %f , Jstim = %f
\n',Voltage,Ik,Ix,INa,ICa,Jstim);
    %printf('Gates: Xgate = %f, mgate = %f, hgate = %f, jgate = %f, dgate
        = %f,
fgate = %f \n',
Gates(1,1),Gates(1,2),Gates(1,3),Gates(1,4),Gates(1,5),Gates(1,6));
    %printf('Calcium = %f \n',Gates(counter,7));

    voltagearray(counter)=Voltage;
    timearray(counter)=ctime;
    Ikarray(counter)=Ik;
    Ixarray(counter)=Ix;

    counter = counter +1;

    ctime = ctime + timestep;
end


figure(1);

subplot(2,2,1);
plot(timearray,Ikarray,'r'); hold on;
plot(timearray,Ixarray,'g'); hold off;

title('Potassium Currents');
xlabel('Time t [msec]');
ylabel('Current I [micro ampse]');
legend('Ik','Ix','Location','NorthEastOutside');

subplot(2,2,2);
plot(timearray,Gates(:,2)); hold on;
plot(timearray,Gates(:,3),'r'); hold on;
plot(timearray,Gates(:,4),'g'); hold off ;

title('Sodium Activation Gates');
xlabel('Time t [msec]');
ylabel('Activation Gates');
legend('m','h','j','Location','NorthWestOutside');

subplot(2,2,3);
```

```
plot(timearray,Gates(:,5)); hold on;
plot(timearray,Gates(:,6),'r'); hold off;

title('Calcium Activation Gates');
xlabel('Time t [msec]');
ylabel('Activation Gates');
legend('d','f','Location','NorthWestOutside');

subplot(2,2,4);
plot(timearray,voltagearray);
title('Transmembrane Action Potential');
xlabel('Time t [msec]');
ylabel('Potential V [volts]');

saveas(1,"brfigures.png");

pause();


 function current = Kcurrent(V)

% Solve current value for k channel using simple fuction
current = 0.35*( 4*(exp(0.04*(V+85))-1) / (exp(0.08*(V+53))+exp(0.04*(V
   +53))) +
(0.2*(V+23)/(1-exp(-0.04*(V+23)))));
endfunction


 function [current,x1] = Xcurrent(V,dt,xgate)

    % Calculate alpha beta values
    alpha_x = alphabeta(V,0.0005,0.083,50.0,0.0,0.0,0.057,1.0);
    beta_x = alphabeta(V,0.0013,-0.06,20.0,0.0,0.0,-0.04,1.0);

    % activation gate initial value
    xnot=(xgate-alpha_x/(alpha_x+beta_x));

    % solve simple ode
    x1=xnot*exp(-(alpha_x+beta_x)*dt) + alpha_x/(alpha_x+beta_x);

    % return current value
    current= (x1*0.8*( exp(0.04*(V+77.0))-1.0)/(exp(0.04*(V+35.0))));

endfunction


 function [current,m,h,j] = Nacurrent(V,dt,mgate,hgate,jgate)

    gna=4;
    gnaca=0.003;
```

```
    ena=50.0;

    % Calculate alpha beta values
    alpha_m =   alphabeta(V,0.0,0.0,47.0,-1.0,47.0,-0.1,-1.0);
    beta_m  =   alphabeta(V,40.0,-0.056,72.0,0.0,0.0,0.0,0.0);
    alpha_h =   alphabeta(V,0.0126,-0.25,77.0,0.0,0.0,0.0,0.0);
    beta_h  =   alphabeta(V,1.7,0.0,22.5,0.0,0.0,-0.082,1.0);
    alpha_j =   alphabeta(V,0.055,-0.25,78.0,0.0,0.0,-0.2,1.0);
    beta_j  =   alphabeta(V,0.3,0.0,32.0,0.0,0.0,-0.1,1.0);

    % activation gate initial value
    mnot=(mgate-alpha_m/(alpha_m+beta_m));
    hnot=(hgate-alpha_h/(alpha_h+beta_h));
    jnot=(jgate-alpha_j/(alpha_j+beta_j));

    % solve simple ode
    m=mnot*exp(-(alpha_m+beta_m)*dt) + alpha_m/(alpha_m+beta_m);
    h=hnot*exp(-(alpha_h+beta_h)*dt) + alpha_h/(alpha_h+beta_h);
    j=jnot*exp(-(alpha_j+beta_j)*dt) + alpha_j/(alpha_j+beta_j);

    % return current value
    current=  (gna*m^3*h*j + gnaca)*(V-ena);
endfunction


 function [current,d,f,Ca] = Cacurrent(V,dt,dgate,fgate,Ca)

    steps=5;
    h=dt/steps;

    gca=0.09;

    % Calculate alpha beta values
    alpha_d =   alphabeta(V,0.095,-0.01,-5.0,0.0,0.0,-0.072,1.0);
    beta_d  =   alphabeta(V,0.07,-0.017,44.0,0.0,0.0,0.05,1.0);
    alpha_f =   alphabeta(V,0.012,-0.008,28.0,0.0,0.0,0.15,1.0);
    beta_f  =   alphabeta(V,0.0065,-0.02,30.0,0.0,0.0,-0.2,1.0);

    % activation gate initial value
    dnot=(dgate-alpha_d/(alpha_d+beta_d));
    fnot=(fgate-alpha_f/(alpha_f+beta_f));

    % solve simple ode
    d=dnot*exp(-(alpha_d+beta_d)*dt) + alpha_d/(alpha_d+beta_d);
    f=fnot*exp(-(alpha_f+beta_f)*dt) + alpha_f/(alpha_f+beta_f);


    % Determine calcium levels
    for i=1:1:steps
```

```
        Ca = Ca+
h*(-10^(-7)*gca*d*f*(V+82.3+13.0287*log(Ca))+0.07*(10^(-7)-Ca));
    end

    % return current value
    current = (gca*d*f*(V+82.3+13.0287*log(Ca)));

endfunction


 function ab = alphabeta(volt,C1,C2,C3,C4,C5,C6,C7)

ab= (C1*exp(C2*(volt+C3)) + C4*(volt+C5))/(exp(C6*(volt+C3))+C7);

endfunction
```

VITA

Andrew Kail was born in Fort Rucker, Alabama on March 15th, 1989. Shortly after he moved to Fort Campbell, Kentucky. He eventually settled in Adams, Tennessee where he then spent the majority of his life. In 2007 he graduated from Rossview High School and went on to study Aerospace engineering at the University of Tennessee in Knoxville.

After graduating with a B.S. in Aerospace Engineering in 2011 Andrew's interest in computational fluid dynamics led him to continue his academics as a graduate student at the University of Tennessee. Upon entering graduate school he became involved in projects with Dr.'s Kwai Wong and A.J. Baker. Andrew then earned a graduate research assistantship with the Joint Institue for Computational Sciences in 2012. In that same year he won "Best Graduate Poster" at XSEDE12 for the poster "A Scalable Software Framework for Thermal Radiation Analysis."

In August 2013 Andrew graduated with a M.S. Degree in Aerospace Engineering and plans to continue working with Dr. Kwai Wong and Dr. Haihang You at JICS.