CHAPTER 2 THE SOFTWARE COMPONENTS

2.1 Introduction

There are many issues to be addressed when using multiple libraries together. The scope of the different components presented is quite vast and their dependencies and usage vary extensively. Understanding the importance and the work involved in tying these together requires a thorough overview of each piece of software. Some theory will be covered as well to explain what the libraries do or why they're necessary.

2.2 Finite Difference Vs. Finite Element

Currently many glacial models, including the one used, that run large scale simulations use finite difference methods in order to solve the partial differential equations involved in the numerical approximations. Finite difference methods are used primarily due to the ease of implementation with fluid dynamics, especially over large areas, and the computations are also less intensive than with finite element methods.

Finite element analysis provides a much more efficient way of calculating the solution with an area that is not of a fixed sized. The difference can be summed up by a quote from Gershenfeld about finite elements: "Instead of discretizing the space in which the problem is posed, this (finite element analysis) discretizes the form of the function used to represent the solution." [Gershenfeld, 1999]

As discussed later in chapter 5 using finite difference methods, both for data and for the calculations, can introduce artifacts and erroneous results if not careful since the domain in ice-sheet modeling can change drastically, but the points at which the approximations are being made do not.

2.3 GLIMMER-CISM

GLIMMER-CISM is a three-dimensional thermo-mechanical ice-sheet model that is open source and is developed to better understand ice dynamics within glaciers and sea-rise consequences. GLIMMER (GENIE Land Ice Model with Multiply-Enabled Regions) was originally developed as part of the GENIE (Grid-ENabled Integrated Earth system model) project in order to be the glacial modeling component of the larger model.[Hagdorn et al., 2008]

Beginning in 2002 GLIMMER was used as the basis for advancing current research of ice-sheet dynamics. The current model, GLIMMER-CISM (Community Ice Sheet Model) has incorporated many new elements beyond the standard SIA (Shallow Ice Approximation) that the original model used. [Wiki, 2010]

GLIMMER-CISM now has a higher-order model for mass-balance based on recent ideas by Pattyn [Pattyn, 2003], some verification tools and test suites, more standardization, intermodel comparisons, and a lot of developers building an infrastructure to share data within the community.

GLIMMER-CISM is written primarily in Fortran and uses NetCDF (Network Common Data Form) files as the input and output to the model as well as configuration files for options, model size, simulation lengths, and many other parameters that can be set. This makes the model very capable and adaptable for both small and large scale simulations.

2.4 PHAML

PHAML stands for "Parallel Hierarchical Adaptive MultiLevel method", and it uses several hp-adaptive methods using triangle meshes to solve elliptical partial differential equations. PHAML incorporates a lot more than just the solving methods though. The software contains parallel libraries to run parts of the simulation over multiple processors, it has OpenGL bindings allowing the user to receive 3D representations of the solutions, and it is very extendible allowing other solvers to be used.

PHAML is designed to solve linear elliptical partial differential equations of the form:

$$-\frac{\partial}{\partial x}\left(c_{xx}\frac{\partial u}{\partial x}\right)-\frac{\partial}{\partial x}\left(c_{xy}\frac{\partial u}{\partial y}\right)-\frac{\partial}{\partial y}\left(c_{yy}\frac{\partial u}{\partial y}\right)+c_x\frac{\partial u}{\partial x}+c_y\frac{\partial u}{\partial y}+c_u u=f\ in\ \Omega \quad (2.1)$$

where c_{xx}, c_{yy}, c_x, c_y and f are functions of x and y, and the domain Ω is a bounded, connected, region in R^2.[Mitchell, 2010c] The boundary conditions can be Dirchlet, natural, or mixed. Dirichlet boundary conditions are of the form:

$$u = g\ on\ \partial\Omega_D \quad (2.2)$$

otherwise the boundary is defined by:

$$\left(c_{xx}\frac{\partial u}{\partial x}+c_{xy}\frac{\partial u}{\partial y}\right)\frac{\partial y}{\partial s}-c_{yy}\frac{\partial u}{\partial y}\frac{\partial x}{\partial s}+c_{bc}u=g\ on\ \partial\Omega_N \quad (2.3)$$

where g and c_{bc} are also functions of x and y and everything else is as defined for the equation (2.1). Here s is some parameter going through some finite range to define the boundary as a function $(x(s), y(s))$. These equations are explained thoroughly in the PHAML documentation as well as all of the different consequences of a chosen boundary condition. [Mitchell, 2010c]

PHAML works by allowing a custom PSLG (Planar Straight Line Graph) or mesh to be passed to the main program, and then it retrieves any information it needs about the solution (or the problem) by means of subroutine callbacks. These are listed in the appendix E.2. These callbacks allow you to specify any of the coefficients as well as functions of the elliptical PDE. This makes the solver very flexible since different types of solutions can be found merely be changing one of the callback. For instance, solving Poisson would have the source subroutine returning the functional value at a given point, but by changing the callback to return zero for all points changes the solution to the Laplacian.

The callbacks also allow for several interesting PDE characteristics to be achieved if desired such as non-homogeneous coefficients, using true solutions, derivative and second derivative based functions, etc.

The mesh geometry can also be generated once the program starts if a functional boundary exists. There are callbacks that allow the program to define what the boundary is while PHAML is executing. This is very beneficial when solving for solutions based on a dataset or a function.

One of the most beneficial aspects of PHAML are that it uses hp-adaptive finite element analysis methods. This allows it to solve PDEs by using piece-wise polynomial approximations that employ elements of variable size (h) and polynomial degree (p). As demonstrated by W. Mitchell this allows hp-FEM methods to not only be more accurate than standard FEM, but that they can also achieve “a convergence rate that is exponential in the number of degrees of freedom.” [William F. Mitchell, 2010]

Having this amount of flexibility also means that PHAML has a lot of functionality that can be tweaked. There are many options for nearly every facet of the solving process. This can be very useful for solutions where a particular type of result is desired. The options themselves will not be covered since they are listed and described in detail within the PHAML documentation.[Mitchell, 2010c]

2.5 Triangle

Triangle is summed up as “a two-dimensional quality mesh generator and Delaunay triangulator”.[Shewchuk, 1996] The software is able to read in files describing a two-dimensional graph and build the data structure while adding useful functionality. The software is also capable of dynamically refining the mesh, adding to the mesh, and providing instant feedback. Triangle can generate “exact Delaunay triangulations, constrained Delaunay triangulations, conforming Delaunay triangulations, Voronoi diagrams, and high-quality triangular meshes”. [Shewchuk, 2010] Most of the algorithms that Triangle depends upon are recent advanced triangulation methods as well as sorting and location algorithms. Some algorithms stemming from this research, such as the robust geometric predicates, are also now used in many other areas of study. [Shewchuk, 1997]

Triangle is not used directly by the interface, but is used as the mesh component of PHAML. The GLIMMER-CISM code uses the Triangle program simply to generate the mesh files needed by PHAML. Since it is needed but not directly used, Triangle is not discussed as thoroughly.

2.5.1 Showme

Triangle also comes with a small mesh viewer called 'showme' which is very useful for looking at the mesh files directly. This program opens the .poly files and allows easy access to the properties of the mesh. This is useful in testing the different generation algorithms by being able to view the output file. The only drawback of the application is that it doesn't show the bmarks associated with each node.

CHAPTER 3 DATA STRUCTURES

3.1 Introduction

Important to any software system are the data structures which guide the flow and storage of the information which the software needs. This is increasingly important when dealing with a modeling simulation where the method in which the data is stored directly affects how it can be used and how useful the data stored is.

3.2 NetCDF

The input and output data format that GLIMMER-CISM uses is NetCDF (network Common Data Form) which is commonly used for modeling applications of this nature. Technically it is "a set of interfaces for array-oriented data access" and is widely used in scientific research due the modularity of the framework and the number of accessible libraries provided for it. [Unidata, 2010]

NetCDF allows for any array-based data to be stored in a single file which can encapsulate several variables, time stepping, and multiple dimensions. These attributes make it an excellent form to store the simulation of an ice-sheet as it changes through time, keeping all the variables associated with the simulation. Although the latest versions of NetCDF are able to store data in formats that aren't simply array-based, the simulation data of GLIMMER-CISM only uses the grid based storage.

3.3 Poly Files

Triangle uses poly (polygon) files to represent a 2-dimensional PSLG (Planar Straight Line Graph). The file format is straightforward for a graph representation. There are minor variations that can be made to the format, but for the purpose of this project the output of the file is as follows.

Table 3.1 Poly File Format

POLY File				
First Line	# Vertices	Dimension(2)	# Attributes(0)	# Bmarks(0 or 1)
Following Lines	Vertex #	X Location	Y Location	Boundary Marker
Next Line	# Edges			# Bmarks(0 or 1)
Following Lines	Edge #	Endpoint 1	Endpoint 2	Boundary Marker
Next Line	# Holes			
Following Lines	Hole #	X Location	Y Location	
Last Line	0			

[Shewchuk, 2010]

The endpoints for edges are the vertex numbers of the vertices listed in the file. Each edge and vertex must have a unique number identifying them. Holes are also allowed in the PSLG, but for the scope of this work, no examples with holes were used.

3.4 Mapping

The transformation between the NetCDF format and the poly one is straightforward. This is simply taking a grid and transforming it into a mesh datatype. For the solver, only the ice-sheet is desired and therefore the entire grid is not used. GLIMMER-CISM stores a mask at each timestep which specifies what each node in the graph represents. This mask tells the user whether it is ice, grounded, ocean, land, a grounding line, or other important features of an ice sheet. These are represented as bits in a byte that are turned on or off. Thus, when generating the mesh each node is checked and if it has ice it can be added to the poly file.

An important aspect of the file is the bmark (boundary marker) which allows the mesh to associate a value with each node specifying if the node is on a boundary or as a general marker for a type of node. Since every edge and node have this boundary marker the mapping sets it equal to the value of the mask so that the functionality built into the mask can be extended to be used on the boundary markers. Thus, mask functions checking for ice, whether it is grounded or floating, etc can now be called passing it the boundary marker on a node to have that information during the PDE analysis.

Each node must have a unique identifier (as outlined in the specifications above) and then edges use these vertex identifiers to specify the two nodes they connect. Since this is a triangular mesh format it is important to add in extra edges so that only triangle sections exist in the PLSG. The standard north/south and east/west edges were applied like a standard grid, and then southwest/northeast edges were added where possible. This left some blocky sections on the sides so southeast/northwest edges are also added to all nodes where the other cross edges didn't exist.

Mapping the data back to NetCDF is also done directly due to the built in functionality in PHAML which allows you to specify a point and get the interpolated data from the mesh there. Thus, all the resulting data can be accessed by some data manipulation and reshaping arrays to go in between the mesh and NetCDF. These procedures are outlined more thoroughly in the following chapter which discusses the additions and files directly.

3.4.1 Examples

Creating an accurate representation is straightforward for a smooth ice sheet. An ideal glacier, as shown in 3.1, can yield a smooth mesh with no anomalies. More intricate glaciers though may have adjacent nodes that are connected even though they shouldn't be. Examples of these types of issues are demonstrated in Greenland (fig 3.2) and Antarctica (fig 3.3).

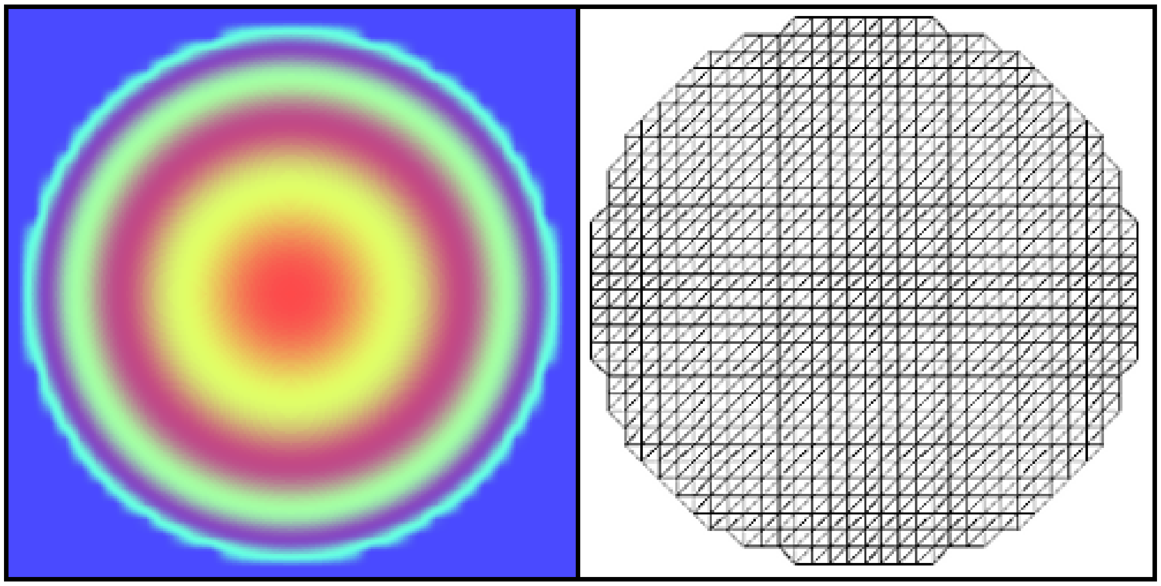

Figure 3.1 A mesh created from the ice thickness of an ice sheet

Greenland has a few anomalies such as the creation of a hole from a large bay area that doesn't actually connect. There are also a couple of islands that are connected to the mainland by an edge. These sort of issues happen because of grid resolution. The larger the distance between nodes of the grid, the more details can be lost in the mesh generation. The mesh of Greenland shown,fig 3.2, is based off of a 15 kilometer grid.

Many resolution based artifacts will also exist around the edges of Antarctica de-

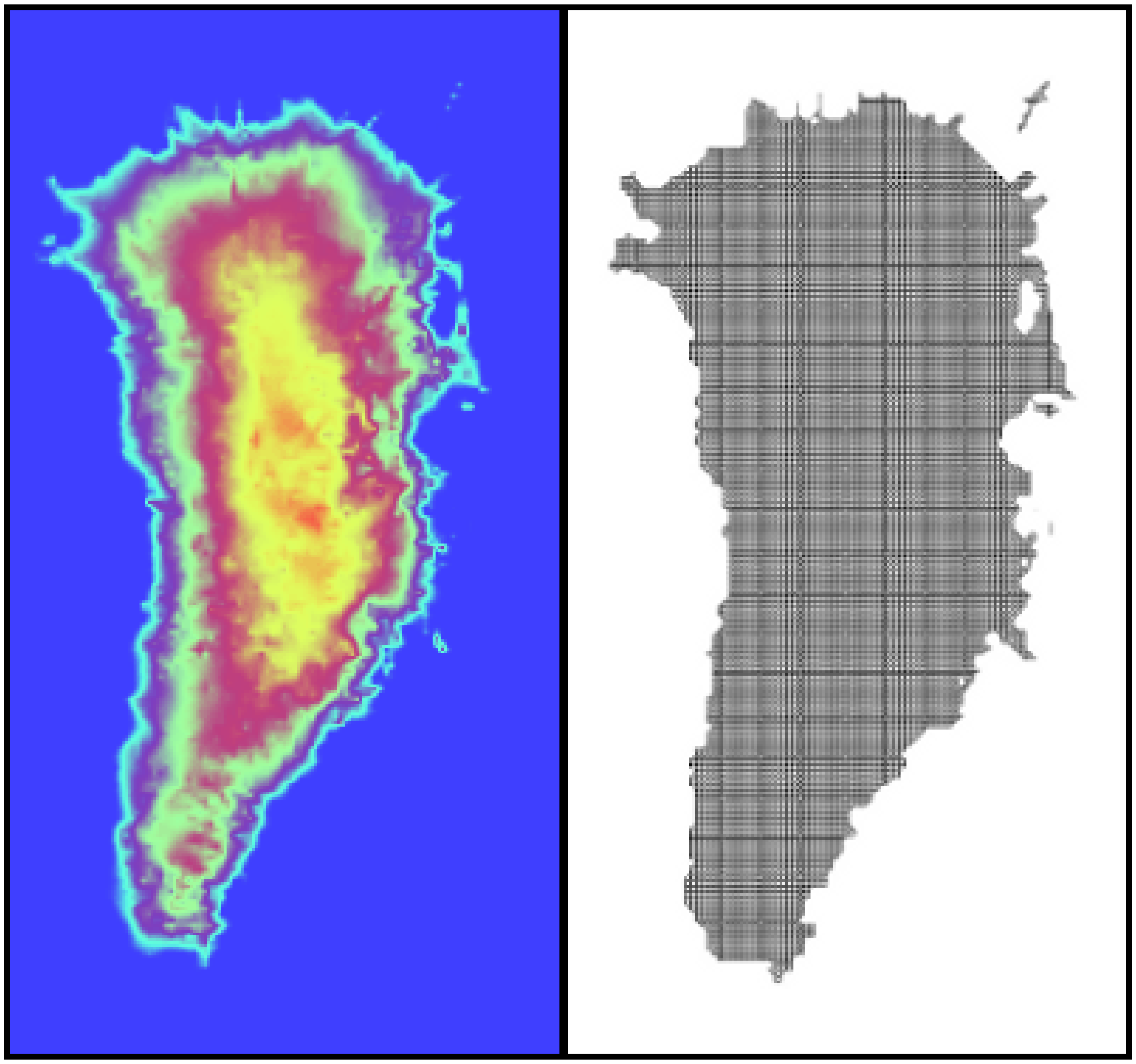

Figure 3.2 Greenland mesh created from the ice thickness

pending on the grid size. Due to the size of Antarctica resolution will always be an issue. The mesh 3.4 was generated from the grid 3.3 which has a 20 kilometer distance between each node. The most noticeable issue is that many of the small islands tend to get lumped together since they're so close together.

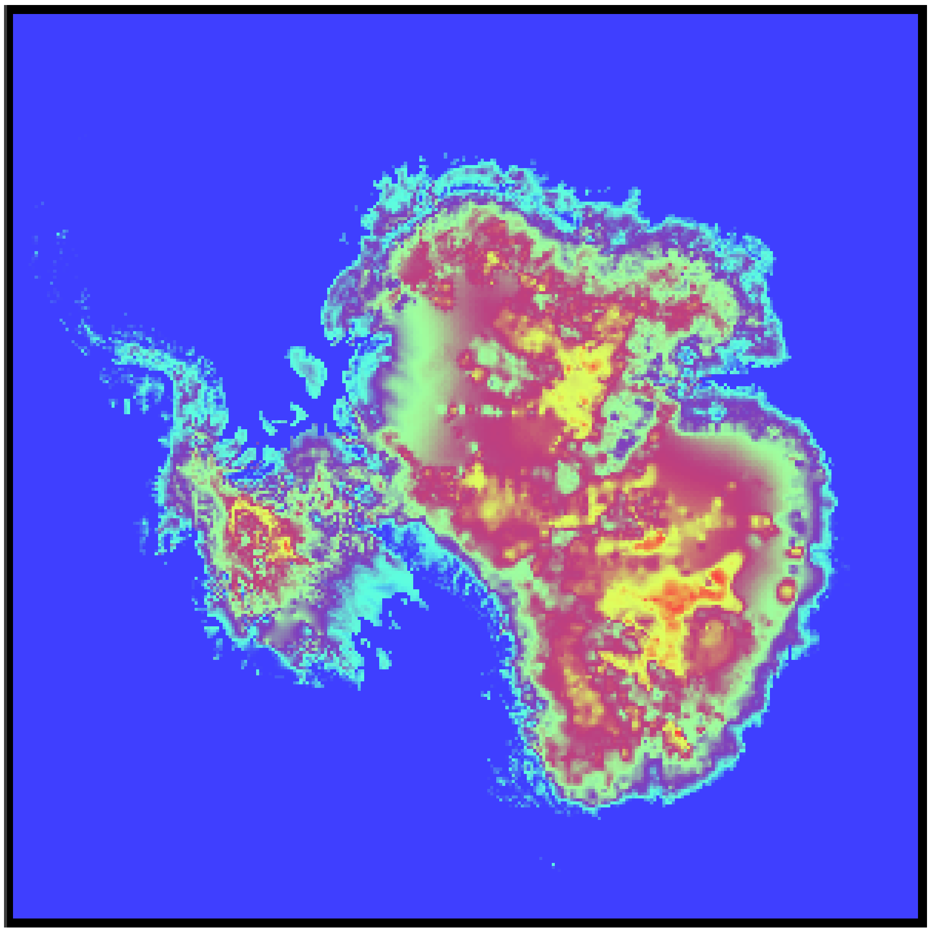

Figure 3.3 Antarctica topological map by ice thickness

3.4.2 Improved Generation Techniques

Clearly shown by the examples 3.2 and 3.4, the mesh generation technique can create unintended anomalies. As mentioned, resolution becomes an issue when trying to

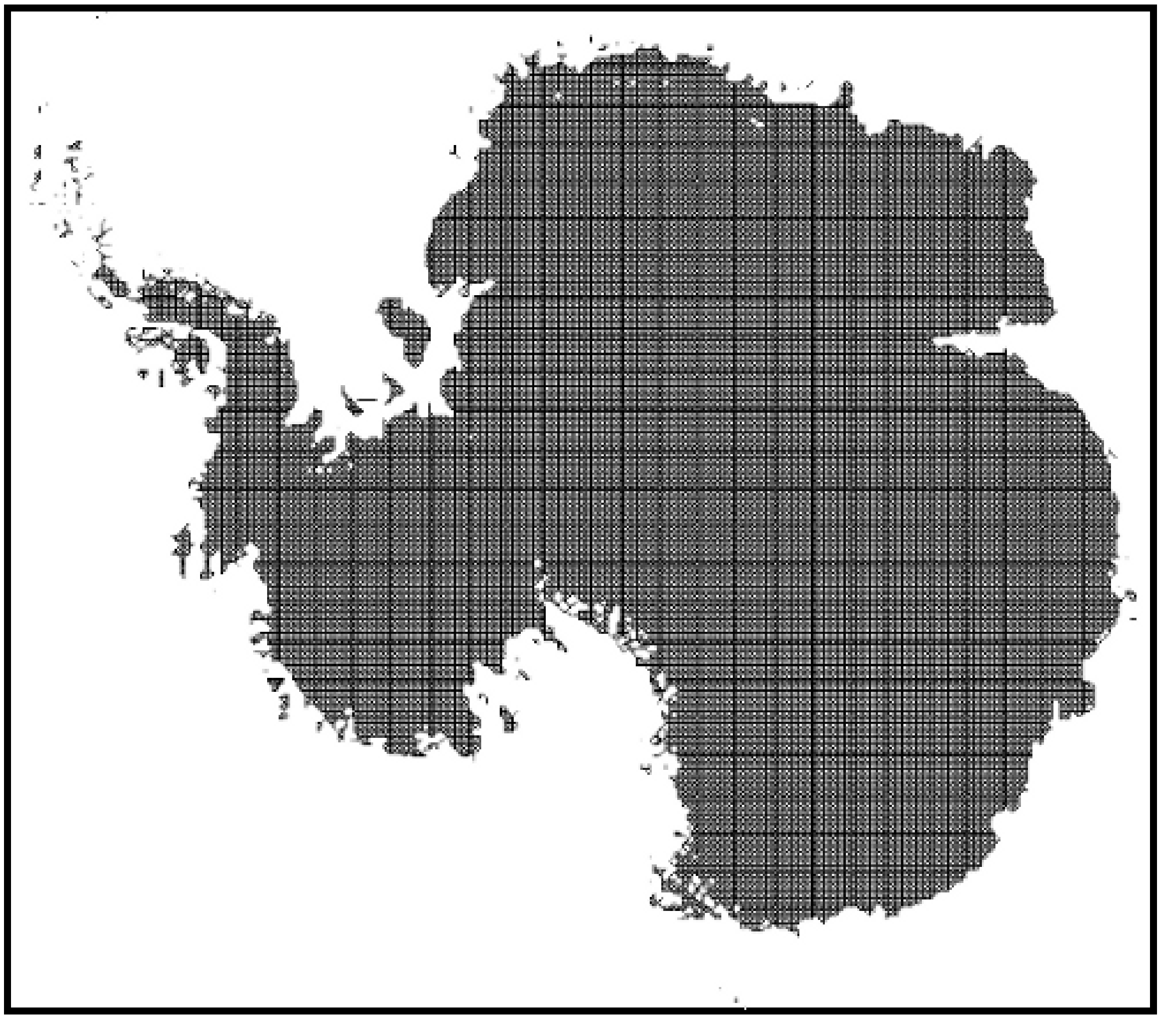

Figure 3.4 Antarctica mesh created from the ice thickness

correctly generate the mesh. Often there are adjacent nodes that are both grounding line nodes but should not be directly connected. This causes the resulting mesh to be larger or irregular in certain areas as small islands become part of the main mesh or jagged grounding lines become smoother. This effect can be lessened by taking a higher resolution grid to begin with so that there is less distance between the nodes. Although this works, these anomalies are always possible and it requires having a higher resolution grid in the areas which are not as important.

For the diagonal edges other approaches were tested as well which check to make sure adjacent edges existed first. These more cumbersome approaches made very little overall difference in the mesh though. A land mass based approach needs to be taken which snakes along the grounding line in order to separate the different islands. This is a tricky and complicated algorithm when working with hundreds or thousands of possible smaller islands. This approach would greatly benefit the overall mesh production though making the simulations and applications more accurate.

Better triangulation checking within the mesh would be very beneficial as well. When there is a misplaced edge or a cycle greater than three Triangle will sometimes fix it, but sometimes they aren't fixed and will cause issues within the simulation.

CHAPTER 4 SOFTWARE INTEGRATION

4.1 Overview

The architecture combining GLIMMER-CISM with PHAML can be partitioned into a few key areas: grid mapping, callback definitions, data manipulation between master and slaves, build systems, debugging techniques, problem definition, and the integration of results into the ice sheet dynamics. Some of these areas are problems beyond the scope of this thesis, but are still worth noting. A large portion of the issues though have to do with the code between the two components and making sure they're working together properly. The resolutions to these issues are discussed in this chapter.

4.1.1 Files

The majority of the code to link the two software components lies in three files:

- phaml_support.F90 - The support file contains the code which does mesh generation, boundary marker assignment, and some helper functions for generating the mesh.
- phaml_pde.F90 - This file acts as a broker between the different problem specific PHAML modules. This becomes the slave process which PHAML will call, and it determines which CISM PHAML module needs to be called.

- phaml_user_mod.F90 - The user module contains data and functions that are needed across all the PHAML callbacks. The implementation and purpose is discussed further in section 4.4.

An example was also created to demonstrate how the PHAML components and support functionality can be used as a module in GLIMMER-CISM. A new module requires two files to be created. The example modules that demonstrates this are located in the following files:

- phaml_example.F90 - Contains PHAML calls with tweaks specific for the problem and any other data or functions needed.

- phaml_example_pde.F90 - This file has all the problem specific PDE callback definitions that will be invoked via phaml_pde.F90.

The example modules must still be called from within GLIMMER-CISM, so a very basic driver is also included which shows how the example modules are used and how the data from GLIMMER-CISM is passed to the modules and vice-versa.

- simple_phaml.F90 - A simple driver demonstrating how to use the example modules and data structures.

In order to work with the GLIMMER-CISM data, PHAML also needs datasets within the standard ice-sheet model data. Thus another custom datatype in CISM has been added to the model called 'phaml' that is used to hold any data PHAML may need during a simulation or any data CISM might want to store from PHAML- this might include initial conditions, flags, boundary conditions, etc. A listing of the datatypes and variable names in this custom type is located in appendix D.

4.1.2 Master/Slaves

PHAML uses subprocesses in order to parallelize the work being done over processors or machines. Because of these 'slaves' PHAML uses MPI in order to communicate back and forth with them. This can cause issues if the PHAML master process does not close correctly. Whenever a PHAML module is used these slave processes will be launched and for a long simulation it is necessary to check to make sure there are no orphaned phaml_slave processes running because they will eat up CPU time and eventually cause some issues with the system.

4.1.3 Function Overview

Following the program can be problematic given the master and slave processes as well as the callbacks. Figure 4.1 shows the basic function call diagram for the overall process beginning with a driver in CISM loading a phaml_module. The colors represent different modules and the dashed lines represent a call made (either direction) between the master and slave. The assumption is also made that functions return data, but this diagram is merely showing the subroutine calls and not data flow.

The chart is read left to right and top to bottom meaning the calls on the first level at the left are executed before the calls at the same level to the right. At each function the subsequent function calls below it are then executed.

The update_usermod subroutine is called multiple times which is explained further in section 4.4. The call 'master_to_slaves' is a subroutine within the master process that the slave calls. The master then returns the usermod data which the master has. In this way whenever the usermod changes in the main process, the changes are propagated down to the slaves.

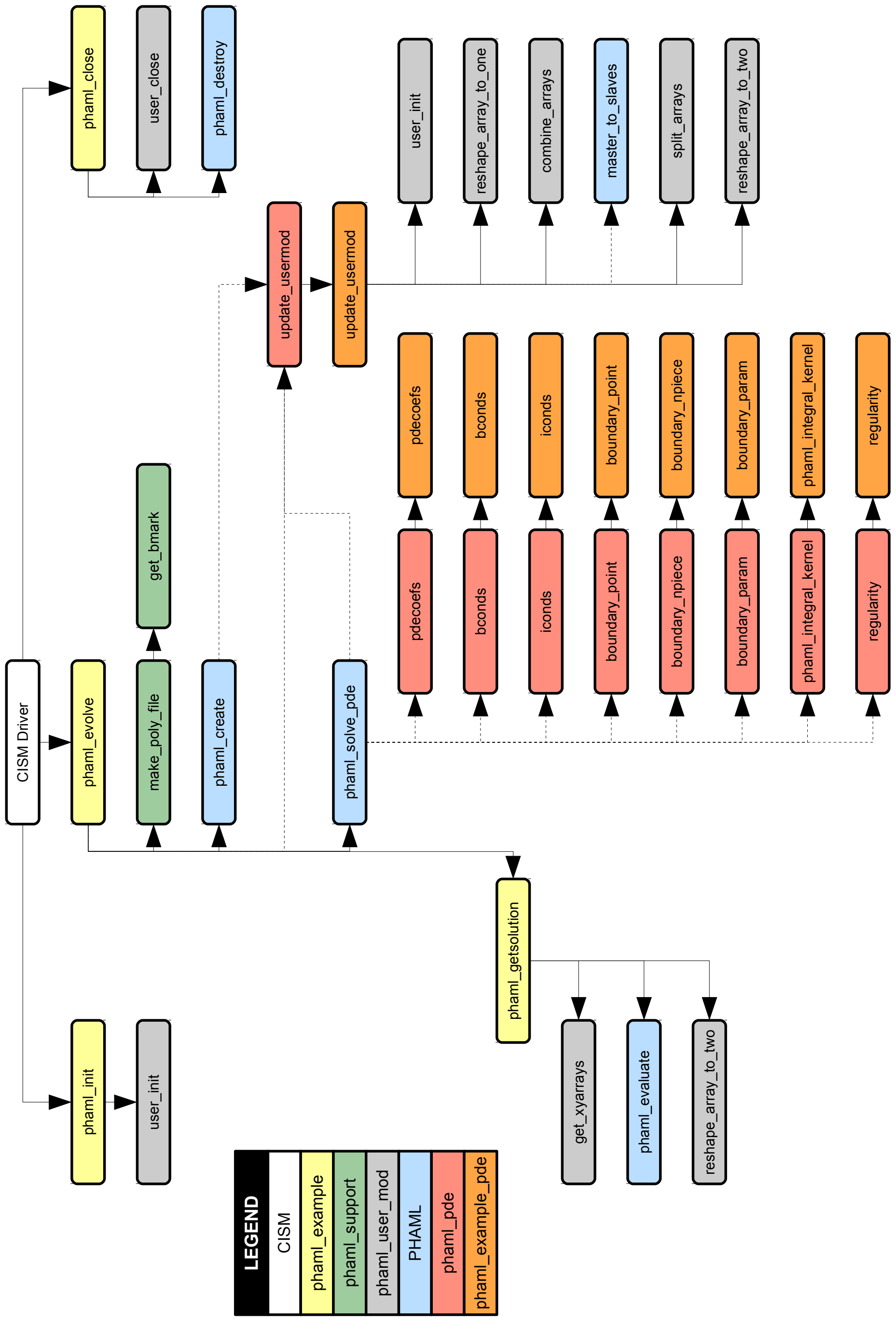

Figure 4.1 An overview of the function calls and how the module functions are connected. Dashed lines indicate a call between master and slave.

4.2 PHAML PDE

The callbacks that PHAML uses are declared in the phaml_pde module. Since the subroutines are problem specific and therefore must be tied to the purpose of the GLIMMER-CISM/PHAML module, the functions defined in phaml_pde act as a broker to find the correct callback for a given module. The names of the callbacks can not be changed because PHAML looks for them explicitly which is why this broker model is necessary. The functions can pass the data to the correct module function by looking at the modnum (module number) declared in the user module

Another important point about the GLIMMER-CISM modules is that since these modules have a specific number associate with them and there is only one user module, only one CISM PHAML module can be in scope at a time. This means that for many computational tasks the CISM PHAML modules will have to be called, run, the results stored, and then discarded until later needed.

4.3 PHAML Support

The PHAML support module contains the mesh generation code that is problem independent and can be used by all PHAML modules that might be created for different applications. The functions are all support functions of the mesh creation in one facet or another. The individual functions are described in Appendix E section E.3.

4.3.1 Mesh Generation

Since PHAML works by solving a partial differential equation on a specific domain, the full GLIMMER-CISM grid (a 2-dimensional matrix) can not be used since part of that domain defines the boundaries and the area around the ice-sheet. This means the grid must be interpreted and a mesh has to be output which defines only the parts of data relevant to the ice-sheet itself. This procedure is detailed more thoroughly in Chapter 3 section 3.4. The code is contained within the support functionality since it doesn't change.

The mesh generation will likely happen multiple times per simulation since upon each time step the glacier is likely to change. This means each time the glacier changes the mesh must be created anew and the simulation will continue with the new mesh of the ice-sheet.

4.4 PHAML Usermod

A very important feature of PHAML that must be handled carefully is the idea of the usermod (user module) that must be declared and maintained. The usermod module is the definition of all variables that might be needed in any of the PHAML callbacks or internal functions. These variables must also be manually passed to the slaves in order for the callbacks to work correctly. Therefore, any callback that might need data from GLIMMER-CISM (initial conditions, boundary conditions, PDE coefficients) must get that information from the user module.

In order to send the information via usermod the data must be set in phaml_user_mod. This process can be tricky when dealing with dynamic arrays so care must be taken. PHAML can only send data to the slaves by passing an integer array and a real array that is one-dimensional. In the case of GLIMMER-CISM 2-dimensional arrays represent the model and are needed in order to set the conditions correctly. Since the arrays are dynamic the master must first send all the size information to the slaves, then call update_usermod again to send the array data. As mentioned the array data must first be reshaped and combined if more than one array is needed, and then passed to the slaves where it is split and then reshaped back to its original form. These transformations must be written specifically for the needed data which is why the user module includes data manipulation functions to ease this transition. These functions are outlined in appendix E section E.4. There may be occasions where problem specific data is needed which is why the update_usermod subroutine actually lies within the phaml_example_pde module, so that it can be tweaked when needed. The general process is shown in figure 4.2.

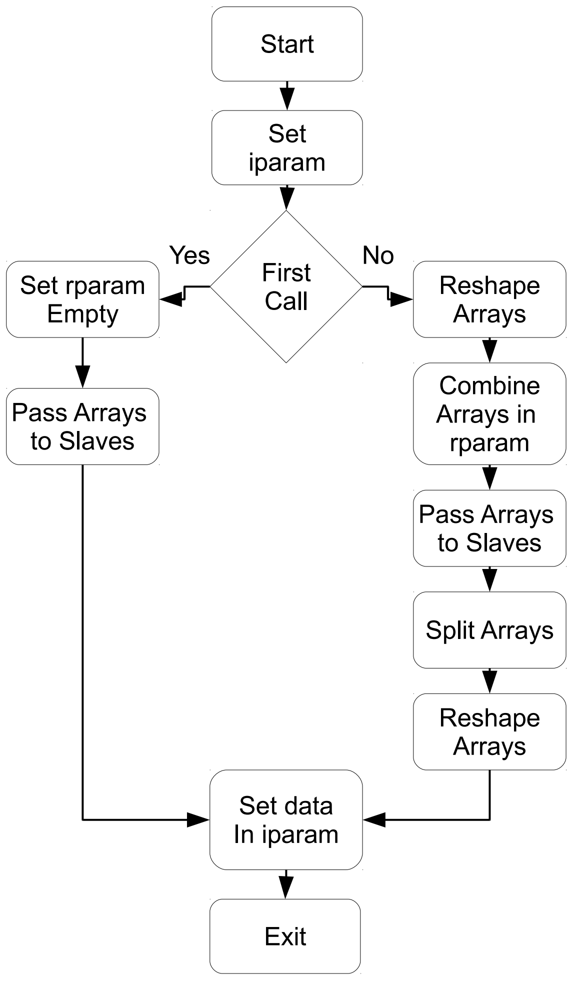

Figure 4.2 The update_usermod flow of instructions to format and pass data to the slave processes.

4.5 PHAML Example Module

The goal of the libphaml addition for GLIMMER-CISM is to provide an interface for PHAML as well as being able to make problem specific drivers. A version might be created to calculate temperature, flux, thickness, or any other number of fields, but each of these problems will model a different PDE and need different data sets in order to be solved. Thus, the name of this file is phaml_example because it is not intended to be a main driver, but merely to illustrate how one is written, what components and functions are needed, and how to use it.

4.5.1 PHAML Example

The file phaml_example.F90 defines the interface for the GLIMMER-CISM code to use the PHAML library. The method in which they're called and the options for the PHAML functions can all be modified if needed based on the purpose of the module. The basic usage is shown in the example module as well as the necessary calls for setting up, maintaining, and closing a PHAML solution, the user module, and all other necessary components. How this module is used in CISM is shown in appendix B

4.5.2 PHAML Example PDE

The phaml_example_pde.F90 file contains all of the callbacks necessary to define the PDE being solved and how it acts at each step. These subroutines define the coefficients, functions, and variables of the PDE 2.1 that PHAML is solving. These functions are all called from the phaml_pde module based on the modnum variable in the user module. The functions themselves are described in section E.2 of the appendix and the user module functions and variables are explained in section E.4.

4.6 Build System

Incorporating one software system as a library of another can create a lot of issues. One of the most daunting ones is the build system. Combining them requires making sure all dependencies get included in the build, that they're built in the correct order, and that certain aspects remain optional in the build process.

Making sure that PHAML remains easy to include, but yet still separate is extremely important to this work. For the majority of simulations needed to be run by GLIMMER-CISM, PHAML will not be needed which means it should not be required to be installed or built along with the project. Having this convenience requires a bit of convolution in the build system. GLIMMER-CISM uses the Autoconf/Make tools for handling the compiling process with all the possible options and dependencies.

The amount of code needed to handle PHAML by itself is not excessive, but it takes quite a bit of time to understand exactly what different parts of the build system are doing. PHAML also requires some extra building procedures since the PHAML specific modules must also be built as a slave process for the master. This added complexity must be handled carefully, and also requires extra scripting in order to build and install the slave separate from the main CISM executables.

The extra dependencies that PHAML requires can be fairly minimal, but can also be quite expansive depending on the features used. The most invasive libraries would be if the OpenGL extensions were desired for debugging or visualization preferences. These are excellent options to have since they are very versatile and provide a unique perspective on how PHAML is working and what the solution is. Including these however, requires a lot of dependency checking, additional compile tags, an additional graphics executable to be built, and can introduce a lot of possibility for build errors.

4.7 Documentation

Documentation is important in any software project, but especially so when interfacing between projects. The difficulty lies in the need to thoroughly understand both pieces of software in order to use them effectively. This requires using the documentation from each project individually, which is why the intermediate documentation is quite necessary. There must be explanation as to how the frameworks as explained in the individual software resources have been modified or mapped in order to work with other tools.

Another issue with documentation is formatting and standardization within the project. GLIMMER-CISM uses Doxygen which is a documentation system that has many advantages: configurable, can generate manual documentation in many formats, and it works with many languages. [Dimitri, 2010] This allows all code added to GLIMMER-CISM to use code documentation as reference, so that code documentation and usage reference can be written only once.

The interface code for this project conforms to the Doxygen standard. The example module does not since these PHAML functions may change greatly for different uses, and so only basic definitions are provided as provided by the PHAML documentation.

BIBLIOGRAPHY

Van Den Berg. Effects of spatial discretization in ice-sheet modelling using the shallow-ice approximation. *Journal of Glaciology*, 52(176):89, 2006.

Tim Bocek. Integration of higher-order physics in the community ice sheet model: Scientific and software concerns. *Unpublished*, pages 59–70, 2009.

Ed Bueler. Exact solutions and verification of numberical models for isothermal ice sheets. 2005.

Ed Bueler. Exact solutions to the thermomechanically coupled shallow-ice approximation: effective tools for verification. *Journal of Glaciology*, 2007.

CISM Community. Glimmer, the community ice sheet model, Accessed May 2010. URL `http://glimmer-cism.berlios.de/`.

L. Demkowicz, J. Kurtz, D. Pardo, W. Rachowicz, M. Paszynski, and A. Zdunek. *Computing with Hp-Adaptive Finite Elements*. Chapman and Hall, CRC Press, 2007.

Dimitri. Doxygen, Accessed May 2010. URL `http://www.stack.nl/~dimitri/doxygen/index.html`.

N. Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, 1999.

M. Hagdorn, I. Rutt, N. R. Hulton, and A. J. Payne. The 'glimmer' community ice sheet model, 2008.

Schoof Pattyn Hindmarsh. Mismip: Marine ice sheet model intercomparison project. 2008. doi: 10.1029/2004GL022024.

P. Huybrechts, T. Payne, and The EISMINT Intercomparison Group. The EISMINT benchmarks for testing ice–sheet models. *Ann. Glaciol.*, 23:1–12, 1996.

IPCC. *Fourth Assessment Report: Climate Change 2007: The AR4 Synthesis Report.* Geneva: IPCC, 2007. URL `http://www.ipcc.ch/ipccreports/ar4-wg1.htm`.

G McGranahan, D Balk, and B Anderson. The rising tide: assessing the risks of climate change and human settlements in low elevation coastal zones. *Environment and Urbanization*, 19(1):17–37, 2007. doi: 10.1177/0956247807076960.

William Mitchell. Phaml, Accessed May 2010a. URL `http://math.nist.gov/phaml/`.

William F. Mitchell. A collection of 2d elliptic problems for testing adaptive algorithms. *NISTIR 7668*, 2010b.

William F. Mitchell. Phaml user's guide, 2010c.

F. Pattyn. A new three-dimensional higher-order thermomechanical ice-sheet model: basic sensitivity, ice-stream development and ice flow across subglacial lakes. *Journal of Geophysical Research (Solid Earth)*, 108(B8):2382, 2003. doi: 10.1029/2002JB002329.

Frank Pattyn. Assessing the ability of numerical ice sheet models to simulate grounding line migration. 2005. doi: 10.1029/2004JF000202.

Frank Pattyn. Role of transition zones in marine ice sheet dynamics. 2006. doi: 10.1029/2005JF000394.

W T Pfeffer, J T Harper, and S Neel. Kinematic constraints on glacier contributions to 21st-century sea-level rise. *Science*, 321(5894): 1340–3, 2008. ISSN 1095-9203. URL `http://www.biomedsearch.com/nih/Kinematic-constraints-glacier-contributions-to/18772435.html`.

David Pollard and Robert M. DeConto. Modelling west antarctic ice sheet growth and collapse through the past five million years. *Nature*, 458:229–232, 2009. doi: 10.1038/nature07809.

John D. Sheehan. Finite element, Accessed June 2010. URL `http://homepages.dias.ie/~js/000_finiteElement.php`.

Jonathan Richard Shewchuk. Triangle, Accessed May 2010. URL `http://www.cs.cmu.edu/~quake/triangle.html`.

Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, October 1997.

Unidata. Unidata, Accessed May 2010. URL `http://www.unidata.ucar.edu/software/netcdf/`.

CISM Community Wiki. Development of a community ice sheet model, Accessed May 2010. URL `http://websrv.cs.umt.edu/isis/index.php/Main_Page`.

Majorie A. MClain William F. Mitchell. A survey of hp-adaptive strategies for elliptic partial differential equations. *Annals of the European Academy of Sciences*, (preprint), 2010.