APPENDIX A SETUP

A.1 Introduction

This appendix presents some compilation notes for the Fortran source files, these can be found in the Glimmer-CISM2 project on the Berlios SVN server. These include the following files: phaml_user_mod.F90, phaml_support.F90, phaml_pde.F90, phaml_example.F90, phaml_example_pde.F90, and simple_phaml.F90.

These instructions all assume a POSIX-compatible system. Although all of the software can be compiled on other operating systems, this has not been attempted on any other OS and therefore is absent. PHAML must be compiled after the graphics libraries if the OpenGL graphics are desired. The graphic libraries are optional though and unnecessary for running the ice sheet model. They are merely present for extra visual output if desired.

All projects require make to build. For the following instructions, the assumption will be made that a global programs directory exists at /usr/bin. If the target system does not have this, then it is necessary to register the installation directory with the system globally so that other programs can find it.

A.2 Triangle

A.2.1 Compiling/Installing

Triangle must be compiled separate from PHAML and installed. This process is straightforward. A program called showme is also compiled and can be installed. It allows you to view the mesh files that Triangle generates which is useful for testing purposes.

```
make
cp triangle /usr/bin
cp showme /usr/bin
```

A.3 PHAML

A.3.1 Getting PHAML

PHAML can be downloaded as an archive from the website Mitchell [2010a], and then unarchived into a directory anywhere on the system.

A.3.2 Compiling PHAML

PHAML is relatively easy to compile if all the library dependencies are satisfied. The list of dependencies as well as instructions for additional libraries are at the end of this section. The PHAML user guide Mitchell [2010c] has an excellent section covering setting up some of these libraries and dependencies as well. The Quickstart guide is a must read before attempting any serious PHAML work. A lot more detail is also provided on individual software packages that can be used and the benefits they can provide. These instructions will now assume that the minimum requirements are met.

First the document 'mkmkfile.sh' needs to be edited in the root directory of the PHAML source folder. The following items must be set correctly: DEFAULT_PHAML_ARCH, DEFAULT_PHAML_OS, DEFAULT_PHAML_F90, DEFAULT_PHAML_C, DEFAULT_PHAML_PARLIB, DEFAULT_PHAML_BLAS, and DEFAULT_PHAML_LAPACK. All other variables can be left to their default value. The file 'mkmkfile.sh' lists the available options for each of these as well. A brief overview of these are in the dependencies section below. Once these variables have all been correctly set the file can be closed and run as a script. As the name suggests, this scripts generates the makefile needed to compile the project with make.

Now just invoke make and it should compile provided things were set correctly.

```
./mkmkfile.sh
make
```

If the code does not compile then either the configuration file is incorrect or a dependency is unsatisfied.

A.3.2.1 PHAML Dependencies

These are required dependencies in order to compile and use PHAML.

- **POSIX compliance** - An operating system must be Unix compatible in order to properly work with PHAML. Ubuntu was used for all test work.
- **Make** - The compile system.
- **A Fortran Compiler** - Most Fortran compilers will work. GFortran was the compiler it was tested against.
- **A C Compiler** - CC or GCC. GCC was used.

- **MPI Library** - An MPI server must be running in order for PHAML to communicate with its subprocesses. Openmpi was chosen for all tests.
- **BLAS** - Compiled from source.
- **LAPACK** - Compiled from source.
- **TRIANGLE** - Compiled from source. Instructions are above.

A.3.2.2 PHAML Additional Libraries

PHAML covers all additional libraries in the user guide. The only extra libraries used in this project were the graphic libraries which have a separate section below with instructions.

- **OpenGL** - The main graphics library.
- **GLUT** - The OpenGL Utility kit.
- **F90GL** - This is a custom interface library so that OpenGL can be called from Fortran.

A.3.3 Installing PHAML

PHAML can be installed anywhere on the system. For a system wide install, using the 'opt' folder is convenient, and will be assumed for the instructions. The PHAML home directory is the top level directory that should have a lib and a modules directory inside of it. Then the global system variables need to be set.

```
export PHAML_OS=linux
export PHAML_HOME=/opt/phaml
```

A.3.4 Including Graphics with PHAML

A.3.4.1 GLUT

GLUT can be downloaded from the official site or a version can be downloaded from PHAML's website Mitchell [2010a] that is guaranteed to work with PHAML. For building this system the one from PHAML was used and it is suggested that this is followed.

Compiling Glut can be somewhat problematic since it depends on OpenGL to compile. Specifically, it was unable to compile on a Ubuntu system with an ATI graphics card because the open source ATI drivers did not include the older SGIX functions which were needed. This is largely a vendor issue since the graphics card manufacturer usually supplies the drivers for the card. When an Nvidia card was used GLUT compiled quickly with no issue since their proprietary drivers included all necessary functions.

The first thing to do is to open Imakefile and remove "test" and "prog" from the SUBDIRS variable. It should only say "SUBDIRS = lib". Then you can run mkmkfiles.imake and compile the project.

```
./mkmkfiles.imake
make
```

Installing GLUT is essential in order to compile F90GL and PHAML with graphics. This includes installing the libraries and the source header files. Where OpenGL is installed on a system varies for each operating system. If the development files for OpenGL are installed on the system, then all the GLUT header files in include/GL should be copied to that OpenGL source directory. On the test system this was /usr/include/GL. The all the compiled GLUT libraries need to be copied into the global dynamic library directory. On most UNIX based systems this is /usr/lib.

Depending on how OpenGL libs are detected the libs might need to also be copied to an X11 directory.

A.3.4.2 F90GL

F90GL can be downloaded from the official site which is the same as PHAML's website Mitchell [2010a]. The package can then be unarchived and compiled anywhere on the system.

Compiling F90GL is not that complicated, but it takes more time to set up than GLUT does. The package includes custom compile scripts based on architecture, operating system, Fortran compiler, and OpenGL version. The file that lists what the abbreviations are for is 'mf_key'. Simply use the file that relates to the intended system and then run make while specifying the system. The other important note is that the script may need to be modified to point to the correct GLUT libraries that were just compiled.

```
make -f mflum2
```

A.4 GLIMMER-CISM

There is fairly good documentation for working with GLIMMER on different platforms and therefore the documentation presented here will be focused on compiling GLIMMER-CISM from the repositories as it was done on the test build.

A.4.1 Compiling GLIMMER-CISM

Compiling GLIMMER is fairly straightforward and relies on the MAKE build system like the other projects have. The dependencies are listed below and should be

satisfied before attempting to compile GLIMMER-CISM itself. Once these are installed GLIMMER-CISM2 can be checked out from the repositories. Community

A.4.1.1 GLIMMER-CISM Dependencies

- **Autoconf** - Tool used in the build system.
- **Make** - The compile system.
- **A Fortran Compiler** - Most Fortran compilers should work. GFortran was the compiler it was tested against.
- **A C Compiler** - CC or GCC. GCC was used.
- **NetCDF** - The libraries for reading a NetCDF file.
- **Python** - Many build scripts and drivers use python scripting.

A.4.1.2 GLIMMER-CISM Additional Libraries

GLIMMER-CISM is a very robust system with the ability to add several different solvers, libraries, and components. They will not be listed here, but more information can be found in the user guide. Hagdorn et al. [2008]

In general compiling GLIMMER-CISM proceeds as such:

- Check out the project GLIMMER-CISM2 using svn or get an archived file of the project for download.
- Open a terminal in the directory of the project.
- Run the following commands:

```
./bootstrap
./configure --with-netcdf=/usr FCFLAGS="-DNO_RESCALE
            -O3 -pedantic-errors -fbounds-check"
make
```

The "–with-netcdf=/usr" can be omitted if the build system can find NetCDF in it's default location on the system. [Community, 2010]

A.4.2 Installing GLIMMER-CISM

Provided everything compiled correctly the build system has a built-in method for installing the binaries on the system. Root privileges may be required.

```
make install
```

A.5 Compiling PHAML With GLIMMER-CISM

There are a few changes to the build system in order to provide the ability for PHAML to compile with GLIMMER-CISM as well as a few additional options that are available to debug the system.

After bootstrapping GLIMMER-CISM the same commands need to be run but now specifying to use phaml, and if debugging the OpenGL graphics. Note that the graphics are not needed and are merely for extra visualization during debugging, but there is a lot of overhead required in getting libraries working.

In order to compile PHAML by itself with GLIMMER-CISM you want:

```
./bootstrap
./configure --with-netcdf=/usr --with-phaml=/opt/phaml
            FCFLAGS="-DNO_RESCALE -O3 -pedantic-errors -fbounds-check"
make
make install
```

In order to include the graphics as well you'll need to add the optional configure tags as well as the libraries to include. The graphics tag needs the location of the F90GL files. The OpenGL libraries should automatically be linked by a system-wide install.

```
./bootstrap
./configure --with-netcdf=/usr --with-phaml=/opt/phaml
            --with-phaml-graphics=/opt/f90gl FCFLAGS="-DNO_RESCALE
            -O3 -pedantic-errors -fbounds-check"
make
make install
```

APPENDIX B GLIMMER-CISM/PHAML USAGE

B.1 Example Code

The example code for PHAML demonstrates calling the phaml_xxxx modules from within the ice-sheet model and how to start new solutions. This is the standard way most of the libraries for GLIMMER-CISM are used. This requires loading everything the model requires and running through the full set of calculations on the ice-sheet. The calls to PHAML will be one small part of the overall simulation process. Therefore, the example code is demonstrative of the smaller piece of code that would be in a larger module.

```
use phaml
use phaml_example
use glide_types
type(phaml_solution_type) :: phaml_solution
type(glide_global_type) :: cism_model

!initialize all variables needed
call phaml_init(cism_model,phaml_solution)

!does the evaluation and places the
!solution in cism_model%phaml%uphaml
call phaml_evolve(cism_model,phaml_solution)

!close and free variables
call phaml_close(phaml_solution)
```

This is an example of when only the solution is desired and no intermediate steps are needed. If a nonlinear or relaxation type of simulation is required then it is possible to walk through the solution one step at a time like the example below. All of the options in PHAML can be tweaked in the evolve procedures of each module if needed.

```
use phaml
use phaml_example
use glide_types
type(phaml_solution_type) :: phaml_solution
type(glide_global_type) :: cism_model

!initialize all variables needed
call phaml_init(cism_model,phaml_solution)

!creates the mesh and sets initial conditions
call phaml_setup(cism_model,phaml_solution)

!looping through timesteps
do while(time .le. model%numerics%tend)

    !copy old solution and do one iteration
    call phaml_nonlin_evolve(cism_model,phaml_solution)

    !get the solution and copy to desired variable
    call phaml_getsolution(phaml_solution, cism_model%phaml%uphaml)
end do

!close and free variables
phaml_close(phaml_solution)
```

B.2 Standalone Code

GLIMMER-CISM provides an excellent framework for doing small scale simulations by using a basic set of libraries. This is a good way to work with PHAML as well since

you can integrate it with GLIMMER-CISM for simple tests without the overhead of the entire model. There is an simple example driver like this included in the libphaml source files aptly named simple_phaml.F90.

B.3 Debugging Options

PHAML provides options to hand running code over to a debugger so that slaves can be monitored separately from the master. These are very useful when handling usermod variables or when using many slaves on different processors.

When calling 'phaml_create' the parameter 'spawn_form=DEBUG_SLAVE' can be passed and then slaves will spawn in a an xterm window with a debugger. This requires compiling with the '-g' flag though, and will default to GDB for the debugger. If a different debugger is desired you can set the 'debug_command' parameter to specify which to use.

APPENDIX C ADDING MODULES/DRIVERS

C.1 Introduction

The build system for GLIMMER-CISM is extensive and great care needs to be taken in order to ensure that everything is compiled correctly. This appendix demonstrates how a new phaml_module.F90 file and phaml_module_pde.F90 file can be added to the build system after they have been created.

C.2 Adding Modules

When adding a module the file 'Makefile.am' in the libphaml directory will need to be edited in order to add the new file to the build process and tell the system the necessary dependencies the module requires. We'll assume the file to be edited is named 'phaml_module.F90'. The first things that will have to happen is adding the library to be created to the 'lib_LTLIBRARIES' variable at the top of the 'Makefile.am' file. It will look like:

```
lib_LTLIBRARIES = ......... libphaml_module.la
```

The next thing that must be added is the compile instructions for the new library. Add a block under the other existing library instructions like this:

```
#new phaml library xxxx to be used by glimmer
libphaml_module_la_SOURCES = phaml_module.F90
libphaml_module_la_LIBADD = libphaml_user_mod.la libphaml_support.la \
    #add additional libraries needed here
```

Make sure libraries are not being added multiple times. If another library uses this module not all libraries will need to be added again. If the new library uses any GLIMMER or GLIDE libraries those don't need to be added here because they are included when the binary is built.

And finally in order for the pde callbacks to be available the file phaml_module_pde.F90 will need to be added to the phaml_slave_SOURCES variable as well. The module will also need to be added to the phaml_pde.F90 functions for the module to be used. This requires adding a 'use' statement in each subroutine like so:

```
use phaml_module_pde
```

C.3 Adding Drivers

Sometimes it might be desired to add a driver as a very simplistic version of the GLIMMER-CISM model where only certain portions of the model are used in order to test a new PHAML module. This process is outlined by the 'simple_phaml' binary that is built. Like the libraries, the binary must first be added to the list of binaries to create. The assumption is made that there exists a file phaml_driver.F90 being used for this program.

```
bin_PROGRAMS = ........ phaml_driver
```

Now the binary can be defined by what source files as well as what libraries it needs in order to compile. The order in which the libraries are listed is important. If library A is needed by library B, then the order must be A,B in the library listing.

```
phaml_driver_SOURCES = phaml_driver.F90
phaml_driver_LDADD = $(ac_cv_phaml_prefix)/lib/libphaml.a \
    $(top_builddir)/libglide/libglide.la \
    $(top_builddir)/libglimmer-solve/libglimmer-solve.la \
    $(top_builddir)/libglimmer/libglimmer-IO.la \
    $(top_builddir)/libglimmer/libglimmer.la \
    $(NETCDF_LDFLAGS) $(NETCDF_LIBS) $(MPILIBS) \
    libphaml_user_mod.la libphaml_example.la libphaml_pde.la \
    libphaml_module.la
    #add additional libraries needed here
```

That is everything needed in the Makefile.am file. Now CISM must be rebuilt starting with the bootstrap and configured with the "–with-phaml" option.

APPENDIX D PHAML NETCDF VARIABLES

These variables are contained within the glide_phaml custom type that resides within the glimmer model global type. The load identifier is whether or not GLIMMER-CISM can load this variable from a NetCDF input file.

Table D.1 PHAML Type Variables

PHAML PDE Functions		
Variable Name	Description	Load?
uphaml	The true solution if known	Y
init_phaml	Initial conditions of the PDE domain	Y
rs_phaml	The source values if static	N

APPENDIX E LIBPHAML FUNCTIONS

This appendix describes the functions that are provided in the phaml_example module and that must be maintained in any new modules. The subroutines exist within the example module rather than the support module so that they can be modified or tweaked based on the specific problem. PHAML provides many options to all function calls and this method allows the options to be different between modules if desired without affecting another module.

These are wrappers to ease the use of PHAML and to setup all initial conditions and make sure everything is properly handled with PHAML and the other modules needed. Please see the guide for the native PHAML functions. [Mitchell, 2010c]

E.1 Main Module

- **phaml_init** - This subroutine simply sets the needed variables for the usermod module to work. It does not instantiate the phaml_solution.
- **phaml_setup** - If doing a non-linear PDE problem then this initializes the phaml_solution, creates the mesh, and sets the initial conditions. To solve phaml_nonlin_evolve must be called.
- **phaml_evolve** - This is a single pass solve where the function assumes the problem is linear. It creates the mesh, initializes PHAML, solves the problem,

retrieves the solution, then closes PHAML.

- **phaml_nonlin_evolve** - This subroutine assumes phaml_setup has already been called and that the solution is incremental. It copies the old solution then does another iteration and returns.

- **phaml_getsolution** - Given the phaml_solution variable it simply returns the current solution at the node points of the GLIMMER-CISM model grid.

- **phaml_close** - This destroys the phaml session variable as well as deallocates the variables used in usermod.

E.2 PHAML Callbacks

This section lists the subroutines and functions that PHAML relies on and must be present in order for it to define the PDE and to find a solution. Given the type of solution desired, many of them don't need to be used, but they must all still exist even if returning zero. Please refer to the manual for more detailed descriptions, special circumstances, arguments, and examples. [Mitchell, 2010c]

- **pdecoefs** - This subroutine returns the coefficient and right hand side of the PDE at the point (x,y).

- **bconds** - This subroutine returns the boundary conditions at the point (x,y).

- **iconds** - This routine returns the initial condition for a time dependent problem at the point (x,y).

- **trues** - This is the true solution of the differential equation at point (x,y), if known.

- **truexs** - This is the x derivative of the true solution of the differential equation at point (x,y), if known.
- **trueys** - This is the y derivative of the true solution of the differential equation at point (x,y), if known.
- **boundary_point** - This routine defines the boundary of the domain at the point (x,y) if no mesh was provided.
- **boundary_npiece** - This routine gives the number of pieces in the boundary definition if no mesh was provided.
- **boundary_param** - This routine gives the range of parameter values for each piece of the boundary if no mesh was provided.
- **phaml_integral_kernel** - This is the identity function that PHAML requires and shouldn't need any modification.
- **regularity** - Provides the *a priori* knowledge about the singular nature of the solution if applicable.
- **update_usermod** - This routine updates the module variables by sending them from the master to the slave processes. This function is very important for the simulation to work correctly, and the data formatting is addressed in more detail in section E.4.

E.3 Support Module

These subroutines are independent of PHAML and are simply support functions needed by the various PHAML modules that can be created. There is a possibility

that some of them may need to be modified depending on a particular simulation need, but in general should be applicable to most situations.

- **is_ice_edge** - This function uses the mask to determine if a node is the last node on the glacier to have ice. It does this by checking all four surrounding nodes to make sure at least one of them doesn't have ice.

- **get_bmark** - This function returns the boundary marker required in the .poly file for edges. Currently it uses the 'mask' value, but can be changed depending on other needs.

- **make_ice_poly_file** - This subroutine generates the mesh file that PHAML loads by only using nodes that have ice as decided by the mask in CISM. It uses the get_bmark and is_ice_edge subroutines. Once it writes out the .poly file it calls Triangle in order to process it for use by PHAML.

- **make_full_poly_file** - This subroutine generates the mesh file that PHAML loads by using the full domain space. A rectangular grid will always be output. The function writes out the .poly file and then calls Triangle in order to process it for use by PHAML.

E.4 Usermod Module

The usermod module is a set of variables and routines that are used or might be used from within the PHAML callbacks. Any data coming from GLIMMER-CISM would need to be set in one of these variables and then could be used in a callback. All the data must be passed on to PHAML's slaves though the function 'update_usermod' that is in the set of PHAML callbacks. These callbacks can be tricky and are explained further in section E.2. The usermod module is addressed in chapter 4 section 4.4.

E.4.1 The functions

- **user_init** - This function sets up the initial data for the usermod module.
- **user_close** - This function serves to call any closing subroutines or deallocate any user data that was initialized.
- **array_init** - Once the slaves have the data from user_init, the array variables can be allocated and sent as well. This function should be called from within update_usermod.
- **array_close** - This is to deallocate the arrays used in the usermod function which had to be dynamically allocated.
- **concat_arrays** - Since the usermod requires all data be passed in one array it might be necessary to pass more than one, and it would be easy to lose track of them. This function concatenates them together in a consistent fashion.
- **split_arrays** - This is the inverse function to concat_arrays. It will split an array back into the original two based on the length specified in the usermod data.
- **reshape_array_to_one** - In order to pass data to the slaves in PHAML, all data must be passed in a single dimension array. This function takes a two dimensional array and converts it to one dimension.
- **reshape_array_to_two** - This is the inverse function to reshape_array_to_one. It takes in the single dimension array and splits it back into two dimensions based on the model 'nsn' and 'ewn'.

- **get_xyarrays** - PHAML returns the solution in one long array, so this function returns two arrays with the corresponding node locations in absolute dimensions to pass to the phaml_evaluate function.
- **getew** - Given an x in absolute coordinates, this divides it by the 'dew' and truncates it into an integer in order to return the nearest ew coordinate for the grid.
- **getns** - Given a y in absolute coordinates, this divides it by the 'dns' and truncates it into an integer in order to return the nearest ns coordinate for the grid.

E.4.2 The variables

- **gnsn** - The number of nodes in the north/south direction of the grid.
- **gewn** - The number of nodes in the east/west direction of the grid.
- **gdns** - The representative distance in meters between each node in the north / south direction.
- **gdew** - The representative distance in meters between each node in the east / west direction.
- **num_arrays** - The number of arrays needed to be passed via update_usermod
- **modnum** - The unique identifier (integer) for this module so that the correct callback functions will be used.

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.