Virtual California User Manual

Version 1.0.0

December 28, 2012

Contents

List of Figures

Part I

Preface

Preface

About This Document

This document is organized into three parts. Part I consists of traditional book front matter, including this preface. Part II begins with an introduction to Pyre and the Pyre-compatible version of CitcomS and their capabilities and proceeds to the details of implementation, including several "cookbooks" of short tutorials. Part III provides appendices and references.

The style of this publication is based on the Apple Publications Style Guide (`developer.apple.com/library/mac/documentation/UserExperience/Conceptual/APStyleGuide/APSG_2009.pdf`), as recommended by Python.org (`www.python.org`). The documentation was produced using LyX (`www.lyx.org`) to facilitate the transformation of files from one format to another. LyX is a document processor that encourages an approach to writing based on the structure of your documents, not their appearance. It is released under a Free Software/Open Source license.

Errors and bug fixes in this manual should be directed to the CIG Mantle Convection Mailing List (`cig-mc@geodynamics.org`).

Who Will Use This Document

This documentation is aimed at two categories of users: scientists who prefer to use prepackaged and specialized analysis tools, and experienced computational Earth scientists. Of the latter, there are likely to be two classes of users: those who just run models, and those who modify the source code. Users who modify the source are likely to have familiarity with scripting, software installation, and programming, but are not necessarily professional programmers.

The manual was written for the usage of Virtual California on a variety of different platforms. Virtual California has run on shared memory computers (Sun, Hewlett-Packard, SGI, and IBM), commercial distributed memory machines (Intel and Cray/SGI), clusters (including machines on the NSF TeraGrid), Linux PCs and Mac OS X desktops.

Citation

Computational Infrastructure for Geodynamics (CIG) is making this source code available to you in the hope that the software will enhance your research in geophysics. The underlying C++ code for the Greens function calculations, stress evolution, simulation framework, and the QuakeLib library and associated Python bindings and testing framework were donated to CIG in December of 2012. A number of individuals have contributed a significant portion of their careers toward the development of Virtual California. It is essential that you recognize these individuals in the normal scientific practice by citing the appropriate peer reviewed papers and making appropriate acknowledgements.

The Virtual California development team asks that you cite both of the following:

- Heien, E.M., Sachs, M.K., "Understanding Long-Term Earthquake Behavior through Simulation," Computing in Science and Engineering, pp. 10-20, Sept.-Oct., 2012

- Sachs, M.K., Heien, E.M., Turcotte, D.L., Yikilmaz, M.B., Rundle, J.B., Kellogg, L.H. ”Virtual California Earthquake Simulator” Seismological Research Letters, November/December 2012, v. 83, p. 973-978, doi:10.1785/0220120052

The developers also request that in your oral presentations and in your paper acknowledgements that you indicate your use of this code, the authors of the code, and CIG (`geodynamics.org`).

Support

Support for this work and researchers was provided by multiple sources. This work was supported by the National Aeronautics and Space Administration (NASA) grant number NNX08AF69G, JPL subcontract number 1291967, and NASA Earth and Space Science Fellowship number NNX11AL92H. Support was also given by the Southern California Earthquake Center (SCEC). SCEC is funded by the National Science Foundation Cooperative Agreement EAR-0529922 and U.S. Geological Survey (USGS) Cooperative Agreement 07HQAG0008. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Conventions

Throughout this documentation, any mention of “username” is meant to indicate the user, meaning you should substitute your account name in its place.

Part II

Chapters

Chapter 1

Introduction

Virtual California is a finite element code designed to solve thermal convection problems relevant to earth's mantle released under the GNU General Public License (see Appendix D on page 81). Written in C++, the code runs on a variety of parallel processing computers, including shared and distributed memory platforms. In an effort to increase the functionality of CitcomS to include greater control during simulations on large parallel systems, the software has been reengineered from previous versions of CitcomS to work with a Python-based modeling framework called Pyre. With Pyre, CitcomS can be dynamically coupled with other CitcomS simulations or with other codes such as SNAC, which solves crustal and lithospheric problems.

1.1 About Virtual California

Virtual California is a boundary element code that performs simulations of fault systems based on stress interactions between fault elements to understand long term statistical behavior. It performs these simulations using a model of faults embedded in an elastic half space with arbitrary dips and rakes.

The code uses a calculation assuming linear stress increase in the long term based on element-element interaction calculations governed by Okada's implementation of Greens functions. During the rupture (earthquake) phase elements may rupture and release stress based on a combination of static and dynamic stress thresholds.

CitcomS is a finite element code written in C that solves for thermo-chemical convection within a spherical shell. It can solve for problems within either a full or a restricted spherical domain. Although the code is capable of solving many different kinds of convection problems using the flexibility of finite elements, there are aspects of CitcomS which make it well-suited for solving problems in which the plate tectonic history is incorporated. Variable viscosity, including temperature-, pressure-, position-, composition-, and stress-dependent viscosity are all possible.

The fundamental basis for the numerical solution of any time-dependent convection problem is the sequential solution of an equation of motion and an energy equation. Convection problems are initially valued with boundary conditions, including all of the problems which are solved with CitcomS. The normal sequence of steps for the solution of convection problems starts with an initial temperature field. First, the momentum equation is solved. The solution of this equation gives us the velocity from which we then solve the advection-diffusion equation, giving us a new temperature. CitcomS uses this interleaved strategy. It is possible to run convection backward in time so as to guess an initial condition for a normal forward running initial and boundary value problem. However, users should be aware that even specialists in mantle convection modeling are just now starting to explore methods in this area and, as such, this is an emerging area of research.

This code uses an iterative solution scheme to solve for the velocity and pressure and, as such, a converged solution cannot be guaranteed. Nested inside the iterative scheme, the code uses either a conjugate gradient solver or a full multi-grid solver to solve the discretized matrix equations.

1.2 History

Virtual California (abbreviated VC) started as a limited simulation model for distributed seismicity on the San Andres and adjacent faults in southern California, developed by Rundle (***cite***) in Fortran. This model included stress accumulation, release and interactions between faults. This model was updated in the early 2000s (***cite***) to include major strike-slip faults in California and was named Virtual California. This model and simulation was used to examine recurrence time statistics on California faults by Rundle (***cite***) and Yakovlev (***cite***), where it was concluded that the return times on a fault is well approximated by a Weibull distribution.

In 2010 VC was rewritten by Eric Heien in C++ to have a more modular simulation framework and add support for parallel simulations using OpenMP and/or MPI. The fault model in VC was also more cleanly separated from the fundamental stress calculation to allow simulation of other fault systems, including the Nankai trough in Japan (***cite***). Additional features were also added, including background seismicity and a branching aftershock sequence (BASS) model simulation of aftershocks (***cite***). VC was also used in an effort by the Southern California Earthquake Center to unify and compare the results from several earthquake simulators (***cite***).

In 2011 and 2012

1.3 About QuakeLib

QuakeLib is a C++ library containing key mathematics, geophysics and I/O functionality related to earthquake simulation and result analysis. QuakeLib is currently distributed with and is used by VC, though it can be compiled and installed by itself on a machine. More specifically, QuakeLib contains 1) functions to read, write and validate fault models and earthquake catalogs in the EqSim format, 2) C++ classes to represent and access these models and catalogs, 3) C++ classes to represent faults and associated fault parameters as well as functionality related to the faults, 4) C++ classes and functions to perform vector mathematics and unit/geographic conversions related to modeling, 5) functions to evaluate stress and displacement fields based on Okada's equations (***cite***) given a rectangular fault or point source.

QuakeLib is also written to support extension to other scripting languages through the use of the Simplified Wrapper and Interface Generator (SWIG) (***URL***). Currently this supports wrapping the QuakeLib library in a Python interface though additional scripting languages can be easily added. The scripting extension allows researchers to write analysis and visualization scripts based on the same equations and data formats as the simulation. The Python interface is also used as the basis for a testing framework to ensure that any changes made to the code do not affect the scientific results and that computations on different platforms yield the same results within a specified tolerance.

1.4 QuakeLib and Virtual California

Pyre provides a simulation framework that includes solver integration and coupling, uniform access to facilities, and integrated visualization. The framework offers a way to add new solvers to CitcomS and to fine-tune CitcomS simulations.

Developers have created Pyre classes for CitcomS to facilitate simulation setup. However, they are not independent classes in a strict sense. They still share the same underlying data structure and their functionality is not divided clearly. CitcomS was not designed to be object-oriented and to make it so would require significant investment of effort with little return. However, the lack of object-oriented features does not hinder the coupling of CitcomS with other solvers.

1.5 Governing Equations

With CitcomS, the mantle is treated as an anelastic, compressible, viscous spherical shell under Truncated Anelastic Liquid Approximation. With these assumptions, thermal convection is governed by the equations for conservation of mass, momentum, and energy:

$$(\rho u_i)_{,i} = 0 \tag{1.1}$$

$$-P_{,i} + \left(\eta(u_{i,j} + u_{j,i} - \frac{2}{3}u_{k,k}\delta_{ij})\right)_{,i} - \delta\rho g \delta_{ir} = 0 \tag{1.2}$$

$$\rho c_P \left(T_{,t} + u_i T_{,i}\right) = \rho c_P \kappa T_{,ii} - \rho\alpha g u_r T + \Phi + \rho\left(Q_{L,t} + u_i Q_{L,i}\right) + \rho H \tag{1.3}$$

where ρ is the density, u is the velocity, P is the dynamic pressure, η is the viscosity, δ_{ij} is the Kroneker delta tensor, $\delta\rho$ is the density anomaly, g is the gravitational acceleration, T is the temperature, c_P is the heat capacity, κ is the thermal diffusivity, α is the thermal expansivity, Φ is the viscous dissipation, Q_L is the latent heat due to phase transitions, and H is the heat production rate. The expression $X_{,y}$ represents the derivative of X with respect to y, where i and j are spatial indices, r is the radial direction, and t is time. With phase transitions and temperature and composition variations, the density anomalies are:

$$\delta\rho = -\alpha\bar{\rho}(T - \bar{T_a}) + \delta\rho_{ph}\Gamma + \delta\rho_{ch}C \tag{1.4}$$

where $\bar{\rho}$ is the radial profile of density, $\bar{T_a}$ is the radial profile of adiabatic temperature, $\delta\rho_{ph}$ is the density jump across a phase change, $\delta\rho_{ch}$ is the density difference between the compositions, Γ is the phase function, and C is the composition. The phase function is defined as:

$$\pi = \bar{\rho}g(1 - r - d_{ph}) - \gamma_{ph}(T - T_{ph}) \tag{1.5}$$

$$\Gamma = \frac{1}{2}\left(1 + \tanh\left(\frac{\pi}{\bar{\rho}g w_{ph}}\right)\right) \tag{1.6}$$

where π is the reduced pressure, d_{ph} and T_{ph} are the ambient depth and temperature of a phase change, γ_{ph} is the Clapeyron slope of a phase change, and w_{ph} is the width of a phase transition.

These equations lead to the following normalization in which primed quantities are nondimensional:

$$\rho = \rho_0\rho^{'} \tag{1.7}$$

$$\alpha = \alpha_0\alpha^{'} \tag{1.8}$$

$$g = g_0 g^{'} \tag{1.9}$$

$$\kappa = \kappa_0\kappa^{'} \tag{1.10}$$

$$\eta = \eta_0\eta^{'} \tag{1.11}$$

$$c_P = c_{P0}c_P^{'} \tag{1.12}$$

$$x_i = R_0 x_i^{'} \tag{1.13}$$

$$u_i = \frac{\kappa_0}{R_0}u_i^{'} \tag{1.14}$$

$$T_0 = \Delta T T_0' \tag{1.15}$$

$$T = \Delta T(T' + T_0') \tag{1.16}$$

$$t = \frac{R_0^2}{\kappa_0}t^{'} \tag{1.17}$$

$$H = \frac{\kappa_0}{R_0^2}c_{P0}\Delta T H^{'} \tag{1.18}$$

$$P = \frac{\eta_0\kappa_0}{R_0^2}P^{'} \tag{1.19}$$

$$d_{ph} = R_0 d_{ph}^{'} \tag{1.20}$$

$$\gamma_{ph} = \frac{\rho_0 g_0 R_0}{\Delta T}\gamma_{ph}^{'} \tag{1.21}$$

where ρ_0 is the reference density, α_0 is the reference thermal expansivity, g_0 is the reference gravity, κ_0 is the reference thermal diffusivity, η_0 is a reference viscosity, c_{P0} is the reference heat capacity, R_0 is the radius of the Earth, T_0 is the temperature at the surface, and ΔT is the temperature drop from the core-mantle boundary (CMB) to the surface. Dropping the primes, the equations become:

$$u_{i,i} + \frac{1}{\bar{\rho}}\frac{d\bar{\rho}}{dr}u_r = 0 \tag{1.22}$$

$$-P_{,i} + \left(\eta(u_{i,j} + u_{j,i} - \frac{2}{3}u_{k,k}\delta_{ij})\right)_{,i} + (Ra\bar{\rho}\alpha T - Ra_b\Gamma - Ra_c C)g\delta_{ir} = 0 \tag{1.23}$$

$$\begin{gathered}\bar{\rho}c_P\left(T_{,t} + u_i T_{,i}\right)\left(1 + 2\Gamma\left(1-\Gamma\right)\tfrac{\gamma_{ph}^2}{d_{ph}}\tfrac{Ra_b}{Ra}Di\left(T + T_0\right)\right) = \bar{\rho}c_P\kappa T_{,ii} \\ -\bar{\rho}\alpha g u_r Di\left(T + T_0\right)\left(1 + 2\Gamma\left(1-\Gamma\right)\tfrac{\gamma_{ph}}{d_{ph}}\tfrac{Ra_b}{Ra}\right) + \tfrac{Di}{Ra}\Phi + \bar{\rho}H\end{gathered} \tag{1.24}$$

where Ra, the thermal Rayleigh number, is defined as:

$$Ra = \frac{\rho_0 g_0 \alpha_0 \Delta T R_0^3}{\eta_0\kappa_0} \tag{1.25}$$

This is not the usual definition of the Raleigh number that is based on layer thickness; it is based on the radius of the Earth R_0. So for mantle convection problems where R_0 is slightly more than twice the layer thickness, our Ra is about a factor of 10 larger than by the usual definition. The phase-change Rayleigh number, Ra_b, the chemical Rayleigh number, Ra_c, the internal heating Rayleigh number, Ra_H, and the dissipation number Di , are defined as:

$$Ra_b = Ra \frac{\delta\rho_{ph}}{\rho_0 \alpha_0 \Delta T} \tag{1.26}$$

$$Ra_c = Ra \frac{\delta\rho_{ch}}{\rho_0 \alpha_0 \Delta T} \tag{1.27}$$

$$Ra_H = RaH \frac{R_0^3 - R_{CMB}^3}{3R_0^3} \tag{1.28}$$

$$Di = \frac{\alpha_0 g_0 R_0}{c_{P0}} \tag{1.29}$$

1.6 Numerical Methods

The governing equations are solved with the finite element method [?]. CitcomS employs an Uzawa algorithm to solve the momentum equation coupled with the incompressibility constraints [?, ?]. The energy equation is solved with a Steamline-Upwind Petrov-Galerkin method [?]. Brick elements are used, such as eight velocity nodes with trilinear shape functions and one constant pressure node for each element. The use of brick elements in 3D (or rectangular elements in 2D) is important for accurately determining the pressure, which controls the dynamic topography, in incompressible Stokes flow [?]. The discrete form of Equations 1.22 and 1.23 may be written in the following matrix form [?]:

$$\left(\mathbf{B}^T + \mathbf{C}\right) u = 0 \tag{1.30}$$

$$\mathbf{A}u + \mathbf{B}p = f \tag{1.31}$$

where $\mathbf{A}$ is the "stiffness" matrix, u is a vector of unknown velocities, $\mathbf{B}$ is the discrete gradient operator, $\mathbf{C}$ is the second term in Equation 1.22, p is a vector of unknown pressures, and f is a vector composed of the body and boundary forces acting on the fluid. The individual entries of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and f are obtained using a standard finite element formulation; see [?] for the explicit entries.

In the incompressible case, $\mathbf{C}$ is zero. Equation 1.31 can be transformed by premultiplying by $\mathbf{B}^T\mathbf{A}^{-1}$ and using Equation 1.30 to eliminate the velocity unknowns:

$$\mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}p = \mathbf{B}^T\mathbf{A}^{-1}f \tag{1.32}$$

This equation is solved using the Uzawa algorithm, an established method for solving the minimization of a dual function [?], which simultaneously yields the velocity field. A conjugate gradient scheme [?, ?] is used for this iteration and forms the basis for the technique used in CitcomS.

In the compressible case, there are two different strategies to solve Equations 1.30 and 1.31. The first strategy is to add another layer of iterations when solving Equation 1.32. The right-hand-side vector is updated by the velocity solution of the previous iteration. This equation can be solved using the same conjugate gradient scheme as the incompressible case.

$$\mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}p^{(i)} = \mathbf{B}^T\mathbf{A}^{-1}f - \mathbf{C}u^{(i-1)} \tag{1.33}$$

The second strategy is to transform Equation 1.31 by premultiplying by $\left(\mathbf{B}^T + \mathbf{C}\right)\mathbf{A}^{-1}$ and using Equation 1.30 to eliminate the velocity unknowns:

$$\left(\mathbf{B}^T + \mathbf{C}\right)\mathbf{A}^{-1}\mathbf{B}p = \left(\mathbf{B}^T + \mathbf{C}\right)\mathbf{A}^{-1}f \tag{1.34}$$

This equation is solved using a bi-conjugate gradient stabilized scheme.

1.7 Meshes and Geometry

There are two forms of meshes and geometries for CitcomS. By default CitcomS will produce a mesh within a regional geometry that is bound by lines of constant latitude and longitude. There is an option to generate a global mesh of a spherical shell.

For a regional mesh, CitcomS uses meshes that are regular, although considerable flexibility exists for grid refinement in the regional models. There is an option for mesh refinement in which the mesh is refined as a function of latitude, longitude, or radius. Such refinement is suitable for higher resolutions near boundary layers or within the center of the map domain, but is incapable of increasing the resolution near a curvilinear feature, as a plate boundary, unless that plate boundary is orientated north-south or east-west.

In regional meshes, *theta* (or x) is the colatitude measured from the north pole, *fi* (or y) is the east longitude, and z is the radius. *theta* and *fi* are in units of radians. Figure 1.1 shows the organization of the mesh in a regional problem. The numbering of the nodes is z-direction first, then x-direction, then y-direction. This numbering convention is used for the input and output data.

Figure 1.1: Global Node Numbering. Left: Global node numbering starts at the base of arrow A (theta_min, fi_min, radius_inner), and advances from 1 at the base to *nodez* at the tip. Upon reaching the tip, numbering continues from the base of arrow B (*nodez*+1) to its tip (2 *nodez*), and so on for all nodes on the plane fi = fi_min. Right: After completing each theta radius plane, the fi index is incremented and numbering commences from (theta_min, radius_inner) as on the left.

For a global mesh, CitcomS is also capable of generating a mesh for an entire spherical shell in which elements in map view are approximately equal in area. In the full spherical mode, CitcomS has 12 caps numbered 0 to 11 (Figure 1.2).

Figure 1.2: Topological connectivity of the 12 caps. N is the north pole and S is the south pole. The red line marks the 0 degree meridian.

The caps are approximately square in map view so that the edges of the square are oriented diagonally with respect to latitude and longitude. The four corners of the domain are connected by great circles (Figure 1.3). One would normally associate at least one processor with one cap. However, CitcomS can further decompose the domain such that additional processors are used to divide caps uniformly along the two edges of the caps (Figure 1.3) as well as in radius.

Figure 1.3: Orthographic projection of processors from a full CitcomS mesh in which there are 16 processors in map view for each cap. The CitcomS cap is shown as distinct colors while the processor domains within the caps are indicated by the intensity of the color. This example was produced for a run with 2 processors in radius such that the total number of processors was 12×16×2=384.

Chapter 2

Installation and Getting Help

2.1 Introduction

Virtual California and QuakeLib have been tested on Linux, Mac OS X and several other UNIX based platforms. Virtual California has also been successfully run in parallel on several XSEDE systems and commodity cluster systems. CitcomS has been tested on Linux, Mac OS X, Sun Solaris and several NSF TeraGrid platforms. You should have no trouble installing CitcomS on most Unix-like systems.

Most users will install CitcomS from the released source package. The following sections will lead you through the installation process. Advanced users and software developers may be interested in downloading the latest CitcomS source code directly from the CIG source code repository, instead of using the prepared source package; see Section 2.12 later in this chapter.

2.2 Getting Help

For help, send e-mail to the CIG Mantle Convection Mailing List (`cig-mc@geodynamics.org`). You can subscribe to the Mailing List and view archived discussion at the Geodynamics Mail Lists web page (`geodynamics.org/cig/lists`).

2.3 System Requirements

Installation of CitcomS requires the following:

- A C++ compiler
- An MPI library
- If using Pyre: between Python 2.4 to 2.6, including header files (Python.h), and the pythia package (see below).

You can install all the dependencies in one command on most Linux machines. On Debian, Ubuntu or similar distributions, use this command (as `root`):

```
$ apt-get install build-essential python-dev \
          openmpi-dev openmpi-bin
```

On Red Hat, Fedora, CentOS, OpenSuSE or similar distributions, use this command (as `root`):

```
$ yum install make automake gcc gcc-c++ kernel-devel \
      python python-devel openmpi openmpi-libs openmpi-devel
```

On Mac OS X, Python is installed by default. You will need to install a C compiler and MPI library (see sections below).

MPI installations are typically configured for a particular compiler, and provide a special wrapper command to invoke the right compiler. Therefore, the choice of MPI implementation often determines which C compiler to use.

> **Note:** Users familiar with older versions of CitcomS may prefer to install only the legacy CitcomS tools, `CitcomSFull` and `CitcomSRegional`, and forgo use of Python and the Pyre framework. By default in CitcomS 3.2.0 this is how CitcomS is configured. This process requires only a C compiler and an MPI library. For more information, see Section 2.11 later in this chapter.

If you are going to use solver coupling, you will also need a decent C++ compiler (e.g., g++ 3.2 or newer) and the Exchanger package; see Section 2.7. Optionally, CitcomS can be configured to use a parallel HDF5 library; see Section 2.8 on page 28.

2.3.1 C Compiler

On Unix or Linux systems, there is a high likelihood that a usable C compiler is already installed. To check, type `cc` at the shell prompt:

```
$ cc
cc: no input files
$
```

On Linux, if the `cc` command is not found, install GCC using the package manager for your distribution.

The Mac OS X version of GCC is included in a software development suite called Xcode. Xcode is available as a free download at the Apple Developer Connection (`developer.apple.com`).

> Warning: If you are using an Intel compiler on an Itanium CPU, do not use the `-O3` optimization flag as reports indicate that this optimization level will generate incorrect codes. For any compiler, you should always be careful about the correctness of the compiled codes when using an `-O3` or higher optimization level.

2.3.2 MPI Library

CitcomS requires a library which implements the MPI standard (either version 1 or 2). Several free, open-source implementations of MPI are available.

A popular choice is MPICH (`www-unix.mcs.anl.gov/mpi/mpich`). Other choices include LAM/MPI (`www.lam-mpi.org`) and Open MPI (`www.open-mpi.org`). Installing MPI from source involves walking through the standard GNU build procedure (`configure && make && make install`) while logged in as `root`.

Linux users may have a prebuilt MPI package available for their distribution. On Mac OS X, the Fink package manager offers a prepackaged version of LAM/MPI (`www.lam-mpi.org`); so if you have Fink (`fink.sourceforge.net`) installed, simply enter the following command from a Terminal window to install LAM/MPI:

```
$ fink install lammpi lammpi-dev
```

2.3.2.1 MPI C Compiler Command

Once you have an MPI library installed, make sure its C complier command is on your PATH. Unfortunately, the name of this command varies from one MPI implementation to the next. The CitcomS configuration script searches for the following MPI C command names:

```
mpicc hcc mpcc mpcc_r mpxlc cmpicc
```

2.3.3 Python (Optional)

Your system may already have a suitable Python interpreter installed. To check, type the `python` command:

```
$ python -V
Python 2.5.2
```

Mac OS X 10.3 and later ships with a suitable version of Python preinstalled. If you're using an older version of Mac OS X, or for more information in general, see Python on the Mac (`www.python.org/download/mac`) at the Python web site.

On Linux, simply install the binary system package available for your distribution. Be sure to select the Python development package (typically called `python-dev`) in addition to the core Python package – even if you don't plan on doing any Python software development. The development package contains the Python header files (Python.h), which are necessary for building CitcomS.

If you are working on a cluster and `python` is too old, try poking around a little. Sometimes multiple versions of Python are installed on the same system:

```
$ python -V
Python 2.2.3
$ which python
/usr/bin/python
$ ls /usr/bin/python*
/usr/bin/python /usr/bin/python2 /usr/bin/python2.2
/usr/bin/python24 /usr/bin/python2.4
$
```

In the above scenario, it is useful to create an alias to the newer Python:

```
$ cd ~/bin
$ ln -s /usr/bin/python2.4 python
$ export PATH=$HOME/bin:$PATH
$ cd
$ hash -r
$ which python
~/bin/python
$ python -V
Python 2.4.1
$
```

If absolutely necessary, one can easily build Python from source using a C compiler. You can download Python from the Python website (`www.python.org`). Please note that Pyre in CitcomS does not work with Python 2.7 or higher.

2.3.4 Pythia (Optional)

The current version of pythia package can be downloaded from the CIG Pythia page.

2.4 Downloading and Unpacking Source

To obtain CitcomS, go to the Geodynamics Software Packages web page (`geodynamics.org/cig/software/packages/mc/citcoms`), download the source archive and unpack it using the `tar` command:

```
$ tar xzf CitcomS-3.2.0.tar.gz
```

If you don't have GNU Tar, try the following command instead:

```
$ gunzip -c CitcomS-3.2.0.tar.gz | tar xf -
```

2.5 Installation Procedure

After unpacking the source, use the following procedure to install CitcomS:

1. Navigate (i.e., `cd`) to the directory containing the CitcomS source.

   ```
   $ cd CitcomS-3.2.0
   ```

2. Type `./configure` to configure the package for your system.

   ```
   $ ./configure
   ```

3. Type `make` to build the package.

   ```
   $ make
   ```

If you are content to run CitcomS from the build directory, then you are done. Upon successful completion, the `make` command creates a script called `citcoms` in the `bin` subdirectory; this is the script you will use to run CitcomS. You may wish to add the `bin` directory to your `PATH`.

For more details about `configure`, see Section 2.6 below.

2.5.1 Installing to a Secondary Location (Optional)

Optionally, after building CitcomS, you can install it in a secondary location using the `make install` command. This is not necessary for using CitcomS and is not recommended for most situations. It is documented only for the sake of completeness.

By default, CitcomS is configured to install under `/usr/local`, which is not writable by normal users. To install as an ordinary user, give `make install` the `prefix` option, specifying a directory to which you have write access:

```
$ make install prefix=$HOME/cig
```

The above command will install CitcomS under `$HOME/cig`. Afterwards, you may wish to add `PREFIX/bin` (`$HOME/cig/bin`, in this example) to your `PATH`.

After running `make install`, you may (if desired) run `make clean` in the build directory to save disk space. You are also free to delete the source/build directory altogether. (Note that `make install` installs the examples under `PREFIX/share/CitcomS/examples`.)

2.6 Configuration

The `configure` script checks for various system features. As it runs, it prints messages informing you of which features it is checking for. Upon successful completion, it generates a `Makefile` in each source directory of the package. It also generates a `config.h` header file, which contains system-dependent definitions.

The `configure` script will attempt to guess the correct values of various installation parameters. In the event that the default values used by `configure` are incorrect for your system, or `configure` is unable to guess the value of a certain parameter, you may have to specify the correct value by hand.

> **Important:** If the `configure` script fails, and you don't know what went wrong, examine the log file `config.log`. This file contains a detailed transcript of all the checks `configure` performed. More importantly, it includes the error output (if any) from your compiler. When seeking help for `configure` failures on the CIG Mantle Convection Mailing List (`cig-mc@geodynamics.org`), please send `config.log` as an attachment.

Upon successful completion, `configure` will print a brief configuration summary.

If Pyre is enabled, the `configure` script will automatically check for needed Python dependencies, including the Pythia package (which includes the Pyre framework). If necessary, `configure` will download missing Python packages from the Internet and store them under `deps/`. These are Python packages which are required (either directly or indirectly) by CitcomS.

2.6.1 Configure Usage

For a detailed list of `configure` variables and options, give `configure` the `--help` option:

```
$ ./configure --help
```

The following is a summary of the variables and options that are important when installing CitcomS.

2.6.2 Environment Variables

Environment variables may be specified as arguments to `configure`, e.g.,

```
$ ./configure CC=icc # use the Intel compiler
```

Variable	Description

PYTHON	Python interpreter. This is useful if you have Python installed in a non-standard location, e.g.,
	`./configure \`
	`PYTHON=/opt/python2.5/bin/python`
	By default, `configure` will search for a suitable Python interpreter using your `PATH` environment variable.

CC	C compiler command. This is usually set to the name of an MPI wrapper command, e.g.,
	`./configure \`
	`CC=/opt/mpich-1.2.6/bin/mpicc`
	See Section 2.6.3 for details and examples.

CFLAGS	C compiler flags. This can be set for optimization flags.
CPPFLAGS	C preprocessor flags; e.g., `-I<dir>` if you have headers in a nonstandard directory.
LDFLAGS	Linker flags; e.g., `-L<dir>` if you have libraries in a nonstandard directory.
LIBS	Library flags; e.g., -l<lib> if additional library is needed

2.6.3 MPI Configuration

By default, `configure` will search for a C compiler using your `PATH` environment variable. It prefers MPI wrapper commands (such as `mpicc`) to ordinary compiler commands (such as `cc` or `gcc`). You may specify the compiler command name manually using the `CC` variable:

```
$ ./configure CC=/opt/mpich-1.2.6/bin/mpicc
```

The `configure` script will test for the presence of the MPI header (`mpi.h`) and an MPI library using the C compiler command. If `CC` is set to an MPI wrapper command such as `mpicc`, and/or the MPI header and library files are installed in a standard location (i.e., `/usr/include` and `/usr/lib`), these `configure` tests should succeed without difficulty.

But if CC is set to an ordinary compiler command name (e.g., `cc` or `gcc`) and MPI is installed in a non-standard location, you must manually specify `CPPFLAGS` and `LDFLAGS`, so that the compiler can find the MPI header files and libraries.

2.6.3.1 Manually Specifying MPI `include` and `lib` Directories

```
$ ./configure \
CPPFLAGS="-I/opt/mpich-1.2.6/include" \
LDFLAGS="-L/opt/mpich-1.2.6/lib -lmpich"
```

2.6.3.2 Manually Specifying MPI `include` and `lib` Directories and an Alternative Compiler

```
$ ./configure \
CC=icc \
CPPFLAGS="-I/opt/mpich-1.2.6/include" \
LDFLAGS="-L/opt/mpich-1.2.6/lib -lmpich"
```

Note that it may be necessary to specify the name of the MPI library itself in `LDFLAGS` using the `-l` compiler option. If a library name is not given – or if the given option doesn't work – `configure` will automatically try linking using `-lmpi` and, if that fails, `-lmpich`.

2.7 Exchanger Configuration (Optional)

This version of CitcomS is capable of solver coupling. Two or more instances of CitcomS solvers can be coupled together to solve a problem with different length and time scales [?]. An external Exchanger package is needed for solver coupling. You can download Exchanger at CIG's Exchanger web page (`geodynamics.org/cig/software/packages/cs/exchanger`). After you download and untar the package, you can configure and install Exchanger using these commands:

```
$ cd Exchanger-1.0.1/
$ ./configure --prefix=$HOME/cig
$ make
$ make install
```

By default, CitcomS will attempt to auto-detect your Exchanger installation and will disable Exchanger support if it is not found. You may specify the location of your Exchanger installation by setting the PYTHONPATH environment variable to the appropriate installation prefix.

```
$ cd CitcomS-3.2.0/
$ export PYTHONPATH=$HOME/cig/lib/python2.5/site-packages:$PYTHONPATH
$ ./configure --with-exchanger
```

2.8 HDF5 Configuration (Optional)

For writing its output in binary format, CitcomS requires parallel HDF5 (PHDF5). In turn, PHDF5 requires an MPI library with MPI-IO support and a parallel file system. If an existing installation of the PHDF5 library is not available on your cluster, you can compile it from source by following the instructions in the file `release_docs/INSTALL_parallel` under the HDF5 source tree. Under Debian Linux, you may simply install the `libhdf5-mpich`, `libhdf5-mpich-dev` and `hdf5-tools` packages.

> **NOTE:** Most post-processing and visualization scripts discussed in Chapter 5 only understand ASCII output. An additional step to convert the HDF5 output files to ASCII files in necessary before using those scripts.

By default, CitcomS will attempt to auto-detect your PHDF5 installation, and will disable HDF5 support if it is not found. You may specify the location of your PHDF5 installation by setting the PHDF5_HOME environment variable to the appropriate installation prefix.

```
$ export PHDF5_HOME=/opt/phdf5/1.6.5
$ ./configure --with-hdf5
```

2.8.1 Additional Tools

While the following software is not necessary for the normal operation of CitcomS, you may find it useful for accessing CitcomS data in HDF5 files.

2.8.1.1 NumPy

NumPy is an extension to Python which adds support for multi-dimensional arrays for use in scientific computing. You may download NumPy from the NumPy home page (`numpy.scipy.org`). To compile and install this extension, download it and issue the following commands after extracting it:

```
$ cd numpy-1.0
$ python setup.py install --prefix=$HOME/cig
```

Alternatively, under Debian Linux you can install the `python-numpy` package. On Gentoo Linux, NumPy is available in the `dev-python/numpy` ebuild.

2.8.1.2 PyTables

PyTables is an extension to Python and can expose HDF5 array datasets as Python NumPy arrays. It is available at PyTables (`www.pytables.org`).

To compile and install this extension, download the latest stable version and issue the following commands:

```
$ cd pytables-1.3.3
$ python setup.py install --prefix=$HOME/cig
```

To install on Debian Linux, you may use the `python-tables` package instead. On Gentoo Linux, it is available in the `dev-python/pytables` ebuild.

2.8.1.3 HDFView

HDFView is a visual tool written in Java for browsing and editing HDF5 files. You may download it from the HDFView home page (`hdf.ncsa.uiuc.edu/hdf-java-html/hdfview`).

2.8.1.4 OpenDXutils

In order to import HDF5 files into OpenDX, you need the OpenDXutils package from the Cactus project. Go to the OpenDXutils package website (`www.cactuscode.org/Visualization/openDX`) and follow the instructions to download and install the package. Note that you will need to set both `DXMODULES` and `DXMDF` environment variables before running OpenDX to load the package.

2.9 GGRD Configuration (Optional)

CitcomS can read data from files for initial temperature, boundary conditions, material control and others. By default, these files are in ASCII format, and the data in the files must be ordered in the same way of the node ordering. As a result, whenever the mesh is changed, these data files need to be regenerated to satisfy the ordering requirement. This restriction can be lifted if using GRD data files. GRD files are binary files in NetCDF format (Network Common Data Form). GRD files are widely used in GMT (Generic Mapping Tools) as well.

The GRD file support can be enabled at configuration time. It requires the installation of HC package (`geodynamics.org/cig/software/packages/mc/hc`), a 1D global mantle circulation solver which can compute velocities, tractions, and geoid for simple density distributions and plate velocities. To install HC, you will also need the GMT and NetCDF packages (library and header files). See the `README.TXT` file in HC for instructions on installation. After HC is installed, note the directory (for example: `$HOME/cig/HC-1_0`) of HC installation, and execute these commands:

```
$ export HC_HOME=$HOME/cig/HC-1_0
$ ./configure --with-ggrd
```

You may need to set `GMTHOME` and `NETCDFHOME` environment variables if these packages is not installed in the system directory.

2.10 Batch System Configuration

If you are installing CitcomS on a cluster with a batch system, you can configure Pyre such that the `citcoms` command automatically submits jobs to the batch queue. Pyre contains support for the LSF, PBS, and Globus batch systems.

The command to submit a batch job depends upon the particular batch system used. Further, the command used in a batch script to launch an MPI program varies from one cluster to the next. This command can vary between two clusters, even if the clusters use the same batch system! On some systems, `mpirun` is invoked directly from the batch script. On others, a special wrapper is used instead. If you need help to configure Pyre for your cluster, you can contact CIG Mantle Convection Mailing List (`cig-mc@geodynamics.org`) for assistance.

Properly configured, Pyre can handle job submissions automatically, insulating users from the details of the batch system and the site configuration. This feature has the most value when the system administrator installs a global Pyre configuration file on the cluster (under `/etc/pythia-0.8`), for the benefit of all users and all Pyre-based applications.

For more information on configuring Pyre for your batch system, see CIG's Pythia page (`geodynamics.org/cig/software/packages/cs/pythia/docs/batch`). For more information on batch system configuration as it pertains to running CitcomS, see Section 3.5.2 on page 38.

2.11 Installing with Pyre

By default, CitcomS 3.2.0 does not install using Pyre. To build CitcomS binaries with the Pyre wrappers (necessary for the example Cookbooks), call `configure` using the `--with-pyre` option:

```
$ ./configure --with-pyre=yes
```

The only system requirements for this configuration are an MPI library and a C compiler. With Pyre the make command will build `pycitcoms` and `mpipycitcoms` as well as the script `citcoms` used for running CitcomS. Without Pyre the `make` command will build two command-line tools, `CitcomSFull` and `CitcomSRegional`, for running the full solver and the regional solver, respectively.

2.12 Installing from the Software Repository

The CitcomS source code is available via a Subversion server at the Geodynamics website (`geodynamics.org`). This allows users to view the revision history of the code and check out the most recent development version of the software.

NOTE: If you are content with the prepared source package, you may skip this section.

2.12.1 Tools You Will Need

In addition to the usual system requirements, you will need a handful of additional development tools installed in order to work with the source from the CIG software repository.

First, you must have a Subversion client installed. To check, type `svn`; it should return a usage message.

```
$ svn
Type 'svn help' for usage.
```

For more information on Subversion, visit the Subversion website (`subversion.tigris.org`).

Second, you must have the GNU tools Autoconf, Automake, and Libtool installed. To check, enter the following commands:

```
$ autoconf --version
$ automake --version
$ libtoolize --version
```

For more information about these GNU tools, see the GNU website (`www.gnu.org/software`). The CitcomS v3.2.0 source package was created with Autoconf 2.68, Automake 1.11.2, and Libtool 2.4.2.

2.12.2 Download Source from Subversion

To check out the latest version of the software, use the `svn checkout` command:

```
$ svn checkout http://geodynamics.org/svn/cig/mc/3D/CitcomS/trunk CitcomS
```

This will create the local directory `CitcomS` (if it doesn't already exist) and fill it with the latest CitcomS source from the CIG software repository.

The `CitcomS` directory thus created is called a *working copy.* To merge the latest changes into an existing working copy, use the `svn update` command:

```
$ cd CitcomS
$ svn update
```

This will preserve any local changes you have made to your working copy.

2.12.3 Generating the GNU Build System

Your working directory should now contain a fresh checkout of CitcomS:

```
$ ls
CitcomS
$
```

Before you can run `configure` or `make`, you must generate the necessary files using the GNU tools. The easiest way to do this is to run `autoreconf -i`:

```
$ cd CitcomS
$ autoreconf -i
$ ./configure
$ make
```

The `autoreconf` tool runs Autoconf to generate the `configure` script from `configure.ac`. It also runs Automake to generate `Makefile.in` from `Makefile.am` in each source directory.

The configure script stores dependency information for source files in hidden subdirectories called `.deps`. If source files are added, deleted, moved, or renamed – which may happen if you run `svn update` to merge the latest changes – the `make` command may report errors for source files which no longer exist. If this happens, clean your existing configuration using the `make distclean` command, and then re-run `configure` and `make`:

```
$ make distclean
$ ./configure
$ make
```

The `make distclean` command deletes all files generated by `configure`, including the Makefiles themselves! Therefore, after running `make distclean`, you will not be able to run `make` again until you re-run `configure`.

Chapter 3

Running CitcomS

3.1 Using CitcomS without Pyre

Regardless of whether you build CitcomS with or without Pyre, two binary executables, `CitcomSRegional` and `CitcomSFull`, are always placed under the `bin` directory. These programs are compiled from pure C code and do not use Python or the Pyre framework. Each program has the same usage:

```
$ mpirun [mpi_options] CitcomSRegional inputfile
$ mpirun [mpi_options] CitcomSFull inputfile
```

Two input file examples, one for a regional spherical model and one for a full spherical model, are provided in the `examples/Regional` and `examples/Full` directories, respectively. The meaning of the input parameters is described in Appendix A on page 57.

The pure C version, compared to the Pyre version, shares the same input parameters and functionality, but is less flexible in changing parameters and in launching parallel jobs. Users are encouraged to use the Pyre version if possible. In the following sections and Chapter 6, we will concentrate on the Pyre version only.

3.2 Using CitcomS with Pyre

If you build CitcomS with the Pyre framework, an additional executable, `citcoms`, is placed under the `bin` directory. The `citcoms` executable is a Python script used for running both the regional and full spherical models using Pyre. Executed without any command line options, `citcoms` will run a regional model with default parameters. It can also run a full spherical model if the correct parameters are set (see Section **??** on page ?? for an example).

On input, CitcomS needs numerous parameters to be specified (see Appendix A for a full list). All parameters have sensible default values. Since you will likely want to specify the parameters of your CitcomS runs, you will need to alter both computational details (such as the number of time steps) and controlling parameters specific to your problem (such as the Rayleigh number). These input parameters, or properties in the Pyre terminology, are grouped under several Pyre components.

Most of the properties you will set using CitcomS have names which are identical to the parameters for CitcomS in pure C version, which are described in Appendix A.

3.3 Changing Parameters

There are several methods to set the input parameters for CitcomS: via the command line, or by using a configuration file in `.cfg` format.

3.3.1 Using the Command Line

Pyre uses the following syntax to change properties from the command line. To change the value of a property of a component, use:

```
--[component].[property]=[value]
```

Each component is attached to a facility, so the option above can also be written as:

```
--[facility].[property]=[value]
```

Each facility has a default component attached to it. A different component can be attached to a facility by:

```
--[facility]=[new_component]
```

3.3.2 Using a `.cfg` File

Entering all those parameters via the command line involves the risk of typographical errors, which can lead to undesired results. You may find it easier to write a brief `.cfg` input file that contains the parameters. This file has a format similar to a Windows INI file. The file is composed of one or more sections which are formatted as follows:

```
[CitcomS.component1.component2]
# this is a comment
property1 = value1
property2 = value2  ; this is another comment
```

We strongly recommend that you use `.cfg` files for your work. The files are syntax-colored by the `vim` editor. (Upon termination of each run, all of the parameters are logged in a `.cfg` file.)

3.3.3 Using a `.pml` File

A `.pml` file is an XML file that specifies parameter values in a highly structured format. XML files are intended to be read and written by machines, not edited manually by humans. The `.pml` file format is intended for applications in which CitcomS input files are generated by another program, e.g., a GUI, web application, or a high-level structured editor. This file fomat will not be discussed or used further in the manual. It is composed of nested sections which are formatted as follows:

```
<component name='component1'>
    <component name='component2'>
        <property name='property1'>value1</property>
        <property name='property2'>value2</property>
    </component>
</component>
```

3.3.4 Specification and Placement of Configuration Files

One or more configuration files and command line options may be specified on the command line:

```
$ citcoms a.cfg b.cfg --foo.bar=baz
```

In addition, the Pyre framework searches for configuration files named `CitcomS.cfg` in several predefined locations. You may put settings in any or all of these locations, depending on the scope you want the settings to have:

1. `PREFIX/etc/CitcomS.cfg`, for system-wide settings;
2. `$HOME/.pyre/CitcomS/CitcomS.cfg`, for user settings and preferences;

3. the current directory (`./CitcomS.cfg`), for local overrides.

Parameters given directly on the command line will override any input contained in a configuration file. If more than one configuration files are given on the command line, the later one overrides the former ones. Configuration files given on the command line override all other `CitcomS.cfg` files. The `CitcomS.cfg` files placed in (3) will override those in (2), (2) overrides (1), and (1) overrides only the built-in defaults.

3.4 Uniprocessor Example

CitcomS runs similarly in full spherical or regional modes. For the purpose of this example, you will perform a test run of the regional version on a workstation. Execute the following on the command line:

```
$ citcoms --steps=10 --controller.monitoringFrequency=5 \
--solver.datafile=example0 --solver.mesher.nodex=17 --solver.mesher.nodey=17
```

This runs a default convection problem in a regional domain for 10 time steps and with a mesh of $17 \times 17 \times 9$ nodal points. Since we did not provide the parameter `solver.mesher.nodez`, the default value 9 is used. The model results are written to files `example0.*` with an interval of 5 time steps.

Instead of writing the input parameters on the command line, you can put them in a `.cfg` file. The CitcomS source package contains an `examples` directory (the `make install` command installs the examples under `PREFIX/share/CitcomS/examples`, where `PREFIX` is the `CitcomS` installation directory). In this directory, you will find a configuration file equivalent to the previous example: `example0.cfg`. You can run the model using:

```
$ citcoms example0.cfg
```

3.4.0.1 Example: Uniprocessor, `example0.cfg`

```
[CitcomS]
steps = 5

[CitcomS.controller]
monitoringFrequency = 1

[CitcomS.solver]
datafile = example0

[CitcomS.solver.mesher]
nodex =  17
nodey =  17
```

3.5 Multiprocessor Example

In order to run this example, you should be on a Beowulf cluster with four or more processors, or on a supercomputer; and you should be in the directory in which the input file is located, in this case, the `examples` directory. CitcomS has been extensively used on both environments, using up to several hundred processors. How to run a multiprocessor CitcomS model depends on your hardware and software settings, e.g., whether a batch system is used, what the names of the computers in a cluster are, and how the file system is organized. This section will lead you through the different settings of a parallel environment.

3.5.0.2 Example: Multiprocessor, `example1.cfg`

```
[CitcomS]
steps = 70
```

```
[CitcomS.controller]
monitoringFrequency = 10

[CitcomS.solver]
datafile = example1

[CitcomS.solver.mesher]
nprocx =  2
nprocy =  2
nodex  = 17
nodey  = 17
nodez  =  9
```

This example uses 2 processors in colatitude (x-coordinate), 2 in longitude (y-direction), and 1 in the radial (z-direction), i.e., it uses 4 processors in total. In addition, there will be 17 nodes in x (theta), 17 nodes in y (fi), and 9 nodes in z (radius_inner). The model will run for 70 time steps and the code will output the results every 10 time steps.

It is important to realize that within the example script (and in finite element method, FEM) the term "node" refers to the mesh points defining the corners of the elements. In `example1.cfg`, this is indicated with:

```
nodex  = 17
nodey  = 17
nodez  =  9
```

These quantities refer to the total number of FEM nodes in a given direction for the complete problem, and for the example it works out that within a given processor there will be $9 \times 9 \times 9$ nodes. Note that in the x-direction (or y) that for the entire problem there are 17 nodes and there is one node shared between two processors. This shared node is duplicated in two adjacent processors. Unfortunately, "nodes" sometimes also refer to the individual computers which make up a Beowulf cluster or supercomputer. In the example scripts, this is indicated with:

```
nprocx =  2
nprocy =  2
```

Figure 3.1: Computational Domain. Map view on the configuration of the top layer of the computational nodes and the processors.

3.5.1 Output Directories and Output Formats

CitcomS potentially generates a large number of ASCII files. This means that you will have to organize your directories carefully when running CitcomS so that you can manage these files as well as use a post-processing program contained in this distribution.

How to best manage this large output depends on whether you will use a local file system or a parallel file system. For example, if you have a local hard disk on every machine (node) on a Beowulf cluster, with each hard disk mounted locally to the machine, this scenario is referred to as a local file system in this section. Or you might use some kind of parallel file system on your computer (e.g., NFS, GPFS, PVFS, to name a few), which is mounted on all of the nodes. Usually your home directory is mounted on the parallel file system. The local file system is usually more cost- and time-efficient than the parallel file system.

If you want CitcomS to write its output to the local hard disks, you need to have a common directory structure on all of the local hard disks. For example, if the directory `/scratch` exists on all local hard disks, you can run the example script with:

```
$ citcoms example1.cfg --solver.datadir=/scratch
```

The additional command line option will override the `datadir` property, which specifies the output directory. The output files are then placed in `/scratch` on each individual machine with a filename prefix `example1.`

However, if the output directory name on each local hard disk depends on the machine hostname, you can run the example script with:

```
$ citcoms example1.cfg --solver.datadir=/scratch_%HOSTNAME
```

The special string `%HOSTNAME` will be substituted by the hostname of each machine.

As the final example for a local file system, you can specify an arbitrary output directory for each machine. To do so, you must write a program to be executed on each machine which will print the output directory. The program must be named `citcoms_datadir` and must reside on your path. An example of `citcoms_datadir` can be found in the `visual/` directory. Then you can run the example script with:

```
$ citcoms example1.cfg --solver.datadir=%DATADIR
```

The special string `%DATADIR` will be substituted by the output of `citcoms_datadir` for each machine.

If you want CitcomS to write its output to a parallel file system, you have several choices. You can run the example script as follows (substitute *`username`* with your own username):

```
$ citcoms example1.cfg --solver.datadir=/home/username
```

The output files are then placed in your home directory with a filename prefix `example1`. A potential problem with this approach is that the directory `/home/`*`username`* will be flooded with hundreds of files, perhaps even tens of thousands of files if you are running a model using several tens of processors for thousands of time steps. Alternatively you can have each machine write its output to its own directory, according to its MPI rank. You can run the example script with:

```
$ citcoms example1.cfg --solver.datadir=/home/username/%RANK
```

The special string `%RANK` will be substituted by the MPI rank of each processor. You will see four new directories `/home/`*`username`*`/0`, `/home/`*`username`*`/1`, `/home/`*`username`*`/2`, and `/home/`*`username`*`/3`. The processor of MPI rank 0 will write its output in `/home/`*`username`*`/0` with a filename prefix `example1` (defined by the property `datafile` inside `example1.cfg`) and so on.

The ASCII output can potentially take a lot of disk space. CitcomS can write `gzip` compressed output directly. You can run the example script with:

```
$ citcoms example1.cfg --solver.datadir=/home/username \
--solver.output.output_format=ascii-gz
```

Be warned that the post-process scripts do not understand this output format yet.

The last choice is the most powerful one. Instead of writing many ASCII files, CitcomS can write its results into a single HDF5 (Hierarchical Data Format) file per time step. These HDF5 files take less disk space than all the ASCII files combined and don't require additional post-processing to be visualized in OpenDX. In order to use this feature, you must compile CitcomS with the parallel HDF5 library if you haven't done so already (see Section 2.8 on page 28). You can run the example script with:

```
$ citcoms example1.cfg --solver.datadir=/home/username \
--solver.output.output_format=hdf5
```

The output files will be stored in `/home/`*`username`*`/` with a filename prefix `example1` and a filename suffix `h5`. See Chapter 4 on page 41 for more information on how to work with the HDF5 output.

3.5.2 Launchers and Schedulers

If you have used MPI before, you know that `mpirun` requires several command-line options to launch a parallel job. Or if you have used one of the batch systems, you will know that the batch system requires you to write a script to launch a job. Fortunately, launching a parallel CitcomS job is simplified by Pyre's `launcher` and `scheduler` facilities. Many properties associated with `launcher` and `scheduler` are pertinent to the cluster you are on, and are best customized in a configuration file. Your personal CitcomS configuration file (`~/.pyre/CitcomS/CitcomS.cfg`) is suitable for this purpose. On a cluster, the ideal setup is to install a system-wide configuration file under `/etc/pythia-0.8`, for the benefit of all users.

Pyre's `scheduler` facility is used to specify the type of batch system you are using (if any):

```
[CitcomS]
scheduler = lsf
```

The valid values for `scheduler` are `lsf`, `pbs`, `globus`, and `none`.

Pyre's `launcher` facility is used to specify which MPI implementation you are using:

```
[CitcomS]
launcher = mpich
```

The valid values for `launcher` include `mpich` and `lam-mpi`.

You may find the `dry` option useful while debugging the `launcher` and `scheduler` configuration. To debug the scheduler configuration, use the `--scheduler.dry` option:

```
citcoms --scheduler.dry
```

This option will cause CitcomS to perform a "dry run," dumping the batch script to the console, instead of actually submitting it for execution (the output is only meaningful if you're using a batch system). Likewise, to debug the launcher configuration, use the `--launcher.dry` option:

```
citcoms --launcher.dry
```

This option will cause CitcomS to print the `mpirun` command, instead of actually executing it. (If you're using a batch system, a job will be submitted for execution; when it runs, CitcomS will simply print the `mpirun` command, and the job will immediately terminate.)

3.5.2.1 Running without a Batch System

On a cluster without a batch system, you need to specify on which machines the job will run. Supposing the machines on your cluster are named n001, n002, ..., etc., but you want to run the job on machines n001, n003, n004, and n005 (maybe n002 is down for the moment). To run the example, create a file named `mymachines.cfg` which specifies the machines to use:

```
[CitcomS.launcher]
nodegen = n%03d
nodelist = [1,3-5]
```

The `nodegen` property is a printf-style format string, used in conjunction with `nodelist` to generate the list of machine names. The `nodelist` property is a comma-separated list of machine names in square brackets.

Now, invoke the following:

```
$ citcoms example1.cfg mymachines.cfg
```

This strategy gives you the flexibility to create an assortment of `.cfg` files (with one `.cfg` file for each machine list) which can be easily paired with different parameter files.

If your machine list does not change often, you may find it more convenient to specify default values for `nodegen` and `nodelist` in `~/.pyre/CitcomS/CitcomS.cfg` (which is read automatically). Then, you can run any simulation with no additional arguments:

```
$ citcoms example1.cfg
```

Warning: This assumes your machine list has enough nodes for the simulation in question.

You will notice that a machine file `mpirun.nodes` is generated. It will contain a list of the nodes where CitcomS has run. Save the machine file as it will be useful in the postprocessing step.

3.5.2.2 Using a Batch System

The settings which are important when using a batch system are summarized in the sample configuration file which follows.

```
[CitcomS]
scheduler = lsf    ; the type of the installed batch system

[CitcomS.lsf]
bsub-options = [-a mpich_gm]    ; special options for 'bsub'

[CitcomS.launcher]
command = mpirun.lsf    ; 'mpirun' command to use on our cluster

[CitcomS.job]
queue = normal          ; default queue for jobs
walltime = 5*minute     ; run time limit of the job
```

These settings are usually placed in `~/.pyre/CitcomS/CitcomS.cfg` or in a system-wide configuration file. They can be overridden on the command line, where one typically specifies the job name and the allotted time for the job:

```
$ citcoms example1.cfg --job.queue=debug \
    --job.name=example1 --job.walltime=5*minute
```

The number of nodes to allocate for the job is determined automatically, based upon the simulation parameters.

3.5.3 Monitoring Your Jobs

Once launched, CitcomS will print the progress of the model to the standard error stream (stderr). Usually, the stderr is directed to your terminal so that you can monitor the progress. On some system, the stderr is redirected to a file. In any case, the progress is always saved in a log file (e.g., `example1.log`). The log file contains the convergence progress of the computation and, if an error occurs, debugging output. The time file (e.g., `example1.time`) contains the elapsed model time (in non-dimensional units) and CPU time (in seconds) of every time step. The format of the time file can be found in Appendix C on page 77. The log and time files are output by the rank-0 processor only.

Following your successful run, you will want to retrieve the output files from all the nodes and process them so they can be visualized with the visualization program OpenDX (see Chapter 5 on page 47).

3.6 Using CitcomS on the TeraGrid

The TeraGrid is a set of parallel supercomputer facilities at eight partner sites in the U.S. which creates an integrated, persistent computational resource. Since TeraGrid software is based on commodity clusters, Linux/Unix, and Globus, it should be easier to scale from a laboratory development environment to a high-end environment in a straightforward manner which promotes application performance. Although the TeraGrid is a high-end resource, it was developed to be accessible to the general community of scientists and engineers as a production facility. TeraGrid accounts for small allocations are available directly from CIG for investigators in the U.S.

CitcomS has already been installed and tested on several NSF TeraGrid platforms, including NCSA, SDSC, and TACC. To use CitcomS on these machines, please log in to your TeraGrid account and read the instructions at `$TG_COMMUNITY/CIG/CitcomS/TG_README`. See CIG Community Area Software on the TeraGrid (`geodynamics.org/cig/software/csa/`) for additional information on access and to apply for allocation time.

Chapter 4

Working with CitcomS HDF5 Files

4.1 Introduction

A typical run of CitcomS can create thousands if not millions of ASCII output files. This situation is inefficient since it requires an extra post-processing step for assembling the results from each processor (see Chapter 5 on page 47). Since the v2.1 release, CitcomS solves this problem when running the software on computers that have parallel file systems by assembling a binary HDF5 file in parallel I/O mode. You can skip this chapter if you are not using the HDF5 output format.

4.2 About HDF5

The Hierarchical Data Format (HDF) is a portable file format developed at the National Center for Supercomputing Applications (NCSA) (`hdf.ncsa.uiuc.edu/HDF5`). It is designed for storing, retrieving, analyzing, visualizing, and converting scientific data. The current and most popular version is HDF5, which stores multi-dimensional arrays together with ancillary data in a portable self-describing format. It uses a hierarchical structure that provides application programmers with a host of options for organizing how data is stored in HDF5 files.

HDF5 files are organized in a hierarchical structure, similar to a Unix file system. Two types of primary objects, *groups* and *datasets*, are stored in this structure. A group contains instances of zero or more groups or datasets, while a dataset stores a multi-dimensional array of data elements. Both kinds of objects are accompanied by supporting metadata.

A dataset is physically stored in two parts: a header and a data array. The header contains miscellaneous metadata describing the dataset as well as information that is needed to interpret the array portion of the dataset. Essentially, it includes the name, datatype, dataspace, and storage layout of the dataset. The name is a text string identifying the dataset. The datatype describes the type of the data array elements. The dataspace defines the dimensionality of the dataset, i.e., the size and shape of the multi-dimensional array. The dimensions of a dataset can be either fixed or unlimited (extensible). The storage layout specifies how the data arrays are arranged in the file.

The data array contains the values of the array elements and can be either stored together in a contiguous file space or split into smaller *chunks* stored at any allocated location. Chunks are defined as equally-sized multi-dimensional subarrays (blocks) of the whole data array and each chunk is stored in a separate contiguous file space. Extensible datasets whose dimensions can grow are required to be stored in chunks. One dimension is increased by allocating new chunks at the end of the file to cover the extension.

HDF5 also supports access to portions (or selections) of a dataset by *hyperslabs*, which consist of a subarray or strided subarray of the multi-dimensional dataset. The selection is performed in the file dataspace for the dataset. HDF5 also supports parallel I/O. Parallel access is supported through MPI-IO. The file and datasets are collectively created/opened by all participating processes. Each process accesses part of a dataset by defining its own file dataspace for that dataset. When accessing data, the data transfer property specifies whether each process will perform independent I/O or all processes will perform collective I/O.

4.3 Input Parameters

To enable HDF5 output in CitcomS, all you need to do is include the following section in your `.cfg` input file.

```
[CitcomS.solver.output]
output_format = hdf5
```

Alternatively, you can specify the option `--solver.output.output_format=hdf5` on the command line. The resulting filenames will start with the value of your specified `datafile` input parameter and end with `.h5`.

4.3.1 Optimizing Parallel I/O

There are several platform-dependent parameters used by the HDF5 library and the underlying MPI-IO routines to optimize the performance of parallel I/O. The optimal values for these parameters may vary from file system to file system. Ideally, before compiling CitcomS, the build procedure would configure these parameters based on your platform, but this is not implemented currently.

In order to facilitate the process of gathering I/O performance data from a variety of parallel file systems such as GPFS, PVFS, IBRIX FusionFS, etc., you can specify the following parameters in the CitcomS input file. You may use these parameters to tune the performance of the parallel I/O on your system.

All values are assumed to be specified in bytes, unless otherwise indicated.

1. MPI file hints for collective buffering file access.
   (a) `cb_block_size`: Target nodes will access data in chunks of this size.
   (b) `cb_buffer_size`: Specifies the total buffer space on each target node. Set this parameter to a multiple of `cb_block_size`.
2. HDF5 file access properties.
   (a) `sieve_buf_size`: Maximum size of data sieve buffer.
   (b) `output_alignment`: Alignment interval size in bytes.
   (c) `output_alignment_threshold`: Allocation requests above this size will be aligned on a memory address that is a multiple of `output_alignment`.
   (d) `cache_rdcc_nelmts`: Maximum number of chunks that can be stored in the raw data chunk cache.
   (e) `cache_rdcc_nbytes`: Size of raw data chunk cache in bytes.

For more details, you can refer to the following references:

- **MPI-2: Extensions to the Message-Passing Interface, section 9.2.8** (`www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-2.0/node182.htm`) provides a list of MPI-IO reserved file hints, also available in Section 7.2.8 of the MPI-2 reference book.
- **HDF5 documentation** (`hdf.ncsa.uiuc.edu/HDF5/doc/UG/08_TheFile.html`), Chapter 2, Section 7.3, contains a list of HDF5 file access properties.
- **HDF5 User's Guide: Data Caching** (`hdf.ncsa.uiuc.edu/HDF5/doc/Caching.html`) offers a short explanation of raw data chunk caching.

You should also refer to the documentation for your particular parallel file system and check if there are any other MPI or HDF5 hints that could improve your parallel I/O performance.

Finally, here is an example section that would appear in a typical CitcomS input file:

```
[CitcomS.solver.output]
cb_block_size = 1048576             # 1 MiB
cb_buffer_size = 4194304            # 4 MiB
sieve_buf_size = 1048576            # 1 MiB
output_alignment = 262144           # 256 KiB
output_alignment_threshold = 524288  # 512 KiB
cache_rdcc_nelmts = 521
cache_rdcc_nbytes = 1048576
```

4.4 Data Layout

Time independent data (e.g., node coordinates, grid connectivity) are saved in a file (for example, `test-case.h5`) at the first output stage. Subsequently, each output stage will save time dependent data (e.g., velocity, and temperature) in a separate file (for example, `test-case.100.h5` contains data of time step 100). Most of the output data from CitcomS is specified at the nodes of a logically Cartesian grid and is therefore well represented by multi-dimensional arrays. A cap dimension is defined for addressing each of the CitcomS caps, followed by three spatial indices (i, j, k) in the case of 3D data, two spatial indices (i, j) in the case of 2D surface data, or one spatial index (k) in the case of 1D radial data. An additional dimension is provided for storing the components of vector and tensor data. These data arrays are stored in different groups. Each group has a descriptive name. In addition, there is an `/input` group archiving the input parameters of the model. Two sample `.h5` files are provided in `visual/samples/` directory.

Dataset	**Shape**
`/input`	`N/A`
`/coord`	`(caps, nodex, nodey, nodez, 3)`
`/connectivity`	`(cap_elements, 8)`

Table 4.1: Layout of the time-independent data file

Dataset	**Shape**
`/velocity`	`(caps, nodex, nodey, nodez, 3)`
`/temperature`	`(caps, nodex, nodey, nodez)`
`/viscosity`	`(caps, nodex, nodey, nodez)`
`/pressure`	`(caps, nodex, nodey, nodez)`
`/stress`	`(caps, nodex, nodey, nodez, 6)`
`/geoid`	`(8)` See Section C.6.9 on page 79 for details
`/surf/velocity`	`(caps, nodex, nodey, 2)`
`/surf/heatflux`	`(caps, nodex, nodey)`
`/surf/topography`	`(caps, nodex, nodey)`
`/botm/velocity`	`(caps, nodex, nodey, 2)`
`/botm/heatflux`	`(caps, nodex, nodey)`
`/botm/topography`	`(caps, nodex, nodey)`
`/horiz_avg/temperature`	`(caps, nodez)`
`/horiz_avg/velocity_xy`	`(caps, nodez)`
`/horiz_avg/velocity_z`	`(caps, nodez)`

Table 4.2: Layout of the time-dependent data file

4.5 Accessing Data

As previously indicated, HDF5 is a self-describing binary file format. As such we can use a variety of tools to inspect the structure of an HDF5 file, as well as for retrieving data in any order.

4.5.1 Inspecting Structures

To quickly inspect the structure of an HDF5 file, you can use the command `h5ls` which is included with the HDF5 software:

```
$ h5ls -r file.h5
```

4.5.2 Converting to ASCII Files

You can convert the HDF5 files to the ASCII combined capfiles described in Appendix C.6 for specific time steps by using the command included in CitcomS:

```
$ h5tocap modelname step1 [step2 [...] ]
```

You can also convert the HDF5 files to the velo files described in Appendix C.6.2 on page 78 for restart purposes by using the command included in CitcomS:

```
$ h5tovelo modelname step
```

4.5.3 Accessing Data in Python

The small Python script `h5tocap.py` provides a good example of using the PyTables extension module to access the data contained in the CitcomS HDF5 files. Using PyTables, datasets can be retrieved from disk as NumPy arrays. The retrieval avoids unnecessary copying of data by using hyperslabs, which take advantage of Python's powerful array slice-indexing.

For example, obtaining the node coordinates, temperature, and topography values over the entire surface of the sphere for time step 100 can be done easily with the following code snippet:

```
import tables

h5file = tables.openFile('samples/cookbook1.h5', 'r')
surface_coords = h5file.root.coord[0:12,:,:,-1,:]

data100 = tables.openFile('samples/cookbook1.100.h5', 'r')
surface_temperature = data100.root.temperature[0:12,:,:,-1]
surface_topography = data100.root.surf.topography[0:12,:,:]
```

In this case, the slice `0:12` refers to all caps explicitly, while the empty slice "`:`" refers to the entire extent of the corresponding dimension. The values of `-1` above refer to the last z-index, which corresponds to the location of the surface nodes on each of the caps. Finally, note how both HDF5 datasets and groups are conveniently accessible as Python attributes on the PyTables file object.

For more details, refer to the documentation for PyTables (`www.pytables.org`) and NumPy (`numpy.scipy.org`).

4.5.4 Accessing Data Using HDFView

NCSA HDFView is a visual tool for accessing HDF files. You can use it for viewing the internal file hierarchy in a tree structure, creating new files, adding or deleting groups and datasets, and modifying existing datasets. HDFView is capable of displaying 2D slices of multi-dimensional datasets, with navigation arrow buttons that enable you to range over the entire extent of a third dimension. An example of using HDFView is found in Figure 4.1 which uses data from Cookbook 1 in Section **??** on page ??.

Figure 4.1: A screenshot of HDFView. The left panel shows the hierarchy of the groups and datasets. The right panel shows a 2D slice of a dataset. The bottom panel shows the metadata associated with the selected group or dataset.

Chapter 5

Postprocessing and Graphics

5.1 Introduction

Once you have run CitcomS, you would have a series of output files (potentially spread throughout the file systems of your Beowulf cluster or in a set of directories on your parallel file system). You now have to retrieve and combine the data for the time step (or age) of interest. To visualize your results, it is recommended that you use the open-source Open Visualization Data Explorer, better known as OpenDX. The software is available from the OpenDX website (`www.opendx.org`). If you are using Linux, OpenDX is usually available as package `dx` or `opendx` in your distribution. If you are using Mac OS X, OpenDX is available via Fink (`fink.sourceforge.net`). We provide experimental scripts for a 3D visualization program called MayaVi2; see Section 5.7 on page 51. Scripts using GMT commands to plot 2D cross sections of temperature field are also provided; see Section 5.8 on page 51.

5.2 Postprocessing on a Beowulf Cluster

Generally, the results from your CitcomS run will be distributed on disks attached to individual nodes of your Beowulf cluster. The output files are written in each node under the directory that you specified as the `datadir` property in the input file. To examine those files, log in to a node and change directories to the one you specified with a prefix. For example, if you set `datadir=/scratch/`*`username`* and `datafile=test-case` in your input file, then your output files will be written to `/scratch/`*`username`* and will have the prefix `test-case`.

If you select HDF5 for the output format, the output files will have a `.h5` suffix. You won't need to postprocess the `.h5` files; you can visualize the results using OpenDX. See Section 5.6 on page 50 for more details.

If you select compressed ASCII (`ascii-gz`) for the output format, there is no post-processing script for it. You need to decompress and rename the output files before you can use the post-processing script described below.

If you select ASCII for the output format, you will have many output files. An example of a filename for the velocity output is `test-case.velo.2.10` where `test-case` is the model prefix, `velo` means that this is a velocity data file, `2` corresponds to the processor number (i.e., it is output by the 2nd processor), and `10` corresponds to the time step.

If your run used the time-dependent velocity boundary conditions (`solver.param.file_vbcs=on`), the log file will also have a `current age` line that lists the time-step numbers and their corresponding times. To choose an age to export for postprocessing, you have to determine which time step corresponds to the age that interests you by looking at the log file.

When you execute a CitcomS run, your input parameters will be saved in a file `pidxxxxx.cfg` where `xxxxx` is usually a five-digit number for the process ID. This pidfile contains most of the input parameters, which can be useful for archiving and postprocessing.

The ASCII output files of CitcomS need to be postprocessed before you can perform the visualization.

The script `autocombine.py` can postprocess (retrieve and combine) CitcomS output; it will retrieve CitcomS data to the current directory and combine the output into a few files.

Using `autocombine.py`, retrieve and combine data of time-step 10:

```
$ autocombine.py mpirun.nodes pid12345.cfg 10
```

This reads the MPI machinefile (`mpirun.nodes`) and the CitcomS pidfile (`pid12345.cfg`), then calls other scripts to do the actual job. The general usage for `autocombine.py` is:

```
$ autocombine.py [machinefile] [pidfile] \
[step1] [step2 or more ...]
```

If your Beowulf cluster uses `ssh` (rather than `rsh`) to access the computation nodes, you must manually edit `visual/batchpaste.sh` and replace 'rsh' with 'ssh' in the script.

Once `autocombine.py` has run, you will have 2 files (or 24 files for the full spherical version of CitcomS) formatted as follows:

```
test-case.cap00.10
test-case.cap00.10.general
```

The former file is the data file containing simulation results and is referred to as the "capfile"; its format can be found in Appendix C. The latter file is the OpenDX header for the data.

If you have enabled some `optional_output`, and the optional fields are node-based (e.g, `comp_nd`, `stress`, or `pressure`), these fields will be combined in separated files, with filenames:

```
test-case.opt00.10
test-case.opt00.10.general
```

The former file is the data file containing optional fields and is referred to as the "optfile." Its format is column-based ASCII data. The latter file is the OpenDX header for the data. The header file contains information on the ordering of the optional fields.

5.3 Postprocessing in a Non-Cluster Environment

If you run CitcomS in a non-cluster environment or all of your data can be accessed from the local machine, you can still use `autocombine.py` to combine the data. In this case, the `machinefile` is not needed and can be replaced by `localhost`, such as:

```
$ autocombine.py localhost [pidfile] \
[step1] [step2 or more ...]
```

5.4 Using OpenDX for Regional Sphere Visualization

OpenDX modules designed for CitcomS can be found in the source directory called `visual`. The optional `make install` command installs the OpenDX modules under `PREFIX/share/CitcomS/visual` (where `PREFIX` defaults to `/usr/local`).

In this example, you will use `visRegional.net` to visualize the results of regional CitcomS.

1. Launch OpenDX by typing:

   ```
   $ dx
   ```

   You will see an OpenDX Data Explorer window.

2. Click Edit Visual Programs and select `visRegional.net` from the file selector.

3. You will see a Visual Program Editor window.
4. Select the import tab in the Visual Program Editor's main window and double-click on the FileSelector block, which will open a Control Panel window.
5. In the CitcomSImport filename input box, select the header file of your postprocessed data, e.g.,

   ```
   samples/regtest.cap00.100.general
   ```

6. In the pull-down menu, select Execute ▷ Execute on Change.
7. A new window will appear with the image of the model.
8. If you want to zoom, rotate, change projection, and otherwise manipulate the view of the model, experiment with the menu Options ▷ View Control.

Figure 5.1: Regional Model Visualized with OpenDX. A snapshot of an upwelling (blue isosurface) with a slice of the temperature field (bisecting plane).

Additional options in the control panel window for the regional model include:

- **CitcomSImport reduced** can increase or reduce the resolution of the velocity vectors and grids. Reducing resolution is often necessary for visualizing large dataset interactively. Note that the resolution of other fields (e.g., temperature and viscosity) is not reduced.
- **Core radius** is the size of the orange sphere which represents the core-mantle boundary.
- **Isosurface value** is the temperature isosurfaces. You can change the values of the isosurfaces, including adding additional isosurfaces or deleting existing ones.
- **Slab cut dimension** is the direction of the slab (an OpenDX term for cross-section).
- **Slab cut position** is the position of the slab.

You can change any of the parameters, visualization, or set-up by going back to the main window and clicking on each tab. If you click on each block, you will be able to change the settings for that function.

5.5 Using OpenDX for Full Sphere Visualization

1. After launching OpenDX, you will see an OpenDX Data Explorer window.
2. Click Edit Visual Programs and select `visFull.net` from the file selector.
3. You will see a Visual Program Editor window.
4. Select the import tab in the Visual Program Editor's main window and double-click on the FileSelector block, which will open a Control Panel window.
5. In the Format String of CitcomSFullImport input box, select one of the 12 header files of your postprocessed data, e.g.,

   ```
   samples/fulltest.cap00.100.general
   ```

6. The results of a full spherical CitcomS consist of 12 cap files. In order to import the 12 cap files at the same time, edit the filename with the cap number replaced by printf-styled format string `%02d`, e.g.,

`samples/fulltest.cap%02d.100.general`

7. In the pull-down menu, select Execute ▷ Execute on Change.
8. A new window will appear with the image of the model.
9. If you want to zoom, rotate, change projection, and otherwise manipulate the view of the model, experiment with the menu Options ▷ View Control.

Additional options in the control panel window for the spherical model include:

- **CitcomSFullImport reduced** can increase or reduce the resolution of the velocity vectors and grids. Reducing resolution is often necessary for visualizing large dataset interactively. Note that the resolution of other fields (e.g., temperature and viscosity) is not reduced.
- **Core radius** is the size of the orange sphere which represents the core-mantle boundary.
- **Isosurface value** is the temperature isosurfaces. You can change the values of the isosurfaces, including adding additional isosurfaces or deleting existing ones.
- **Latitude of normal axis** and **Longitude of normal axis** are the directions of the normal axis to the cross-section plane.

5.6 Using OpenDX for HDF5 Visualization

If you use the HDF5 output format (`solver.output.output_format=hdf5`), you can directly visualize the data without postprocessing. First, you need to install and set up the OpenDXutils package (see Section 2.8.1.4 on page 29). Then, open either `visRegional.net` or `visFull.net` in OpenDX.

1. In the pull-down menu, select File ▷ Load Macro, and select the file CitcomSImportHDF5.net to load it.
2. In the Tools panel of the main window, select the CitcomSImportHDF5 module in the Macros category (highlighted in Figure).
3. Place the module in the work space and rewire the network as shown in Figure
4. There are four input tabs in the CitcomSImportHDF5 module.
   (a) The first tab (connected to the left FileSelector module) specifies the HDF5 file containing time-independent information (e.g., `samples/cookbook1.h5`).
   (b) The second tab (connected to the right FileSelector module) specifies the HDF5 file containing time-dependent information (e.g., `samples/cookbook1.100.h5`).
   (c) The third tab (connected to the Integer module) specifies the resolution reduction factor.
   (d) The fourth tab (unconnected, default to 0) specifies which cap(s) to import.
      i. To visualize a regional model, leave this tab unchanged.
      ii. To visualize a full spherical model, double-click this module and change the value of this tab to `{0,1,2,3,4,5,6,7,8,9,10,11}`.
      iii. You can also specify a subset of caps to visualize.
5. After the data are imported, the visualization can be manipulated as described in previous sections.

5.7 Using MayaVi for Visualization

This distribution also comes with scripts for visualizing CitcomS using MayaVi2. MayaVi2 is the successor of MayaVi for 2D/3D scientific data visualization. It is an interactive program allowing elaborate plots of scientific data. MayaVi2 relies on the Visualization Toolkit (VTK) and allows easy scripting in Python. To install MayaVi2, you should follow the instructions on the MayaVi2 website (`https://svn.enthought.com/enthought/wiki/GrabbingAndBuilding`) and on the SciPy website (`www.scipy.org/Cookbook/MayaVi/Installation`). Besides MayaVi2, before using the script you will need to install PyTables (see Section 2.8.1.2 on page 29) and PyVTK; see PyVTK website (`cens.ioc.ee/projects/pyvtk`) for instructions.

The script acts as an extension module of MayaVi2. The installation of the script is still preliminary. You will need to include `PREFIX/share/CitcomS/visual/Mayavi2/citcoms_plugins/` in your `PYTHONPATH` environment variable. You also need to create a directory called `~/.mayavi2` and copy into it the following:

```
PREFIX/share/CitcomS/visual/Mayavi2/mayavi_custom_ui.py
```

To run the script, type this command:

```
$ mayavi2_citcoms_display.py file.step.h5
```

This will launch MayaVi2 and load the data from `file.step.h5`. An example of using this script is found in Figure **??**. You may adjust your visualization pipeline by adding any appropriate VTK filters and modules. For more details about this process, refer to *The Visualization Toolkit User's Guide* (ISBN 1-930934-13-0) or see the VTK website (`www.vtk.org`).

5.8 Using GMT Commands for Visualization

The Generic Mapping Tools (GMT) is a collection of command-line tools for manipulating geographic and Cartesian data that produces PostScript (PS) or Encapsulated PostScript File (EPS) illustrations. GMT is widely used in the geophysics community. Two scripts `plot_layer.py` and `plot_annulus.py`, which can plot the temperature field in horizontal and radial cross sections, respectively, are provided. These scripts use GMT commands to generate EPS images. GMT is very customizable. Users might wish to customize the scripts for their need.

The usage of `plot_layer.py` is:

```
$ plot_layer.py modelname caps step layer
```

The script will look for the capfile(s) `modelname.cap00.step`, if `caps` is 1, or `modelname.cap00.step`, `modelname.cap01.step`, ... through `modelname.cap11.step`, if `caps` is 12. A slice of the capfile at the specified radial layer is then plotted in Mercator projection, if `caps` is 1, or in Hammer-Aitoff projection, if `caps` is 12.

The usage of `plot_annulus.py` is:

```
$ plot_annulus.py modelname step
```

The script will ask you a few questions on how to specify the great circle path. There are three options: (1) a starting point and an azimuth, (2) a starting point and another point on the great circle, and (3) a starting point and a rotation pole. The great circle path is used to construct the radial cross section.

Chapter 6

Cookbooks

6.1 Introduction

These cookbook examples are meant to serve as a guide to some of the types of problems CitcomS can solve. Each of these cookbook problems will only run with CitcomS if Pyre is enabled. Cookbook examples range from regional to full spherical shell problems that address traditional mantle convection problems. These cookbook examples are distributed with the package under the `examples` directory. However, you might need to edit these example scripts slightly to launch the job on your cluster (see Section 3.5 on page 35 for more information).

Cookbooks 1 to 4 introduce the basic parameters and are suitable for all users. Cookbook 5 introduces time-dependent velocity boundary conditions, reading in initial temperature field from files and tuning of the advection solver. Cookbook 6 introduces the pseudo-free-surface formulation for studying short wavelength dynamics topography. Cookbook 7 introduces thermo-chemical convection problem and tuning of the Conjugate Gradient velocity solver. Cookbook 8 introduces compressible convection problem, checkpointing/restarting, geoid and tuning of the Multigrid and Uzawa velocity solver. Cookbook 9 introduces embedding one CitcomS domain within another CitcomS domain, and further tuning of the advection solver. Cookbook 10 introduces how to convert the convection field to seismic velocities and how to generate synthetic seismograms using SPECFEM3D_GLOBE at the CIG Seismology Web Portal.

Part III

Appendices

Appendix A

Input Parameters for CitcomS

A.1 Input Parameters Grouped by Functionality

This section explains the meaning of the input parameters for CitcomS. These parameters are grouped by their functionality. Parameters are given with their default values.

A.1.1 Parameters that Control Input Files

`reference_state=1`
`refstate_file="refstate.dat"`
	If `reference_state` is set to `1`, a simple reference state of $\rho_r = \exp\left((1-r)D_i/\gamma\right)$ is used, with constant gravity, thermal expansivity, and heat capacity. If `reference_state` is set to `0`, the reference state is read from a file `refstate_file`.

`mineral_physics_model=3`	Using which mineral physics model to convert the seismic velocities.

`file_vbcs=off`
`vel_bound_file="bvel.dat"`
	If `file_vbcs` is set to `on`, the top surface velocity boundary conditions are read in from files which have location and name specified by `vel_bound_file`. Requires setting `topvbc=1` to take effect. If you wish to have a uniform top surface velocity boundary condition or some simple geometric pattern, then `file_vbcs` should be set to zero.

`mat_control=off`
`mat_file="mat.dat"`
	If `mat_control` is set to `on`, then the time- and positional-dependent viscosity factor is defined from the files specified by `mat_file`. These parameters allow you to define the material group of each element (such as a moving weak zone). Not working in this version.

`lith_age=off`
`lith_age_file="age.dat"`
`lith_age_time=off`
	If `lith_age` is set to `on`, then the age of each surface nodes is read from the files specified by `lith_age_file`. These parameters control the thermal age of the top thermal boundary condition. If `lith_age_time` is `on`, the files are time-dependent.

A.1.2 Parameters that Control Output Files

`output_format=ascii`	Choose the format and layout of the output files. Can be either `ascii`, `ascii-gz` or `hdf5`. If `ascii-gz` is chosen, the code places gzipped files into `data_dir`, and will put all time-step output into subdirectories of `data_dir`. The same naming logic holds for reading old velo files.
`output_optional="surf, botm,tracer"`	Choose additional output, including `surf`, `botm`, `geoid`, `seismic`, `stress`, `pressure`, `connectivity`, `horiz_avg, tracer, heating, comp_el` and `comp_nd`.
`datadir="."`	Controls the location of output files.
`datafile="regtest"`	Controls the prefix of output file names such as `regtest.xxx`. Cannot contain the "/" character if `output_format=ascii`.
`storage_spacing=10` (in non-Pyre version) or `monitoringFrequency=100` (in Pyre version)	Controls the interval between output files. CitcomS dynamically determines the size of the time step; this means that you might not get an output at the exact time required, but you can always get close depending on how small this number is. Do not make this number too small since outputs slow the code down and you may end up with an unmanageable number of output files.
`checkpointFrequency=100`	The time-step interval between checkpoint output, which can be used later to resume the computation.
`output_ll_max=20`	This parameter controls the maximum degree of spherical harmonics coefficients for geoid output.
`self_gravitation=off`	Considering the effect the self gravitation on the geoid or not.
`use_cbf_topo=off`	Using the Consistent Boundary Flux (CBF) method to compute the dynamic topography or not.

A.1.3 Mesh and Processors Setup

`nproc_surf=1`	This specifies the number of spherical caps of the mesh; must be `1` for regional spherical model and `12` for full spherical model.
`nprocx=1` `nprocy=1` `nprocz=1`	These specify the number of processors in each spherical cap. For a full spherical model, `nprocx` must be equal to `nprocy`
`nodex=9` `nodey=9` `nodez=9`	These specify the number of FEM nodes in each spherical cap. These parameters are not used in the C version if multigrid solver is used. For a full spherical model, `nodex` must be equal to `nodey`
`mgunitx=8` `mgunity=8` `mgunitz=8` `levels=1`	These specify the nested level of multigrid units. Used by multigrid solver only. These parameters are not completely independent to each other. The constraint of Equation **??** must be satisfied. These parameters are not used in the C version if conjgrad solver is used. For a full spherical model, `mgunitx` must be equal to `mgunity`

A.1.4 Domain Size

`theta_min=1.0708`
`theta_max=2.0708`
`fi_min=0`
`fi_max=1`
	These parameters specify the horizontal extent of the computational domain. `theta_min` and `theta_max` are the colatitude measured in radians from the north pole. `fi_min` and `fi_max` are the longitudes measured from the prime meridian eastward in radians. Only in regional CitcomS.

`radius_inner=0.55`
`radius_outer=1.0`
	These parameters specify the radial extent of the computational domain. `radius_inner` and `radius_outer` are the inner and outer radii in non-dimensional units. It is probably more convenient to normalize lengths by the radius of the Earth. If you do this, then the Rayleigh number must be calculated with the radius of the Earth, not the thickness of the mantle. The core mantle boundary is close to having a non-dimensional radius of 0.55.

`coor=0`
`coor_file="coor.dat"`
`coor_refine=0.1,0.15,0.1,0.2`
	If `coor=0`, there will be uniform mesh in the latitudinal, longitudinal, and radial directions.
	If `coor=1`, the coordinate is reading from the file specified by `coor_file`. This is used to have a regular but uneven spacing between elements.
	If `coor=2`, there will be uniform mesh in the latitudinal and longitudinal directions. The mesh in the radial direction is generated according to `coor_refine`, which is a vector of 4 numbers. The 1st value of `coor_refine` specifies the radius fraction of the bottom layer, the 2nd value specifies the fraction of the nodes in the bottom layer, the 3rd value specifies the top layer fraction, and the last value specifies the top layer node fraction.

A.1.5 Restarting the Code

`restart=off`	If `restart` is `on`, each processor will resume the computation from the checkpoint files.
`post_p=off`	Similar to `restart`, except that the model will then only run for 1 time step, which can be useful to regenerate the flow field and calculate the associated observables.

`datadir_old="."`
`datafile_old="regtest"`
`solution_cycles_init=0`
	If `restart` is `on`, for example, processor #5 will read its initial conditions from checkpoint file `regtest.chkpt.5.0` in this case.

A.1.6 Run Length

`minstep=1`
`maxtotstep=1000000`
(only in pure C version)
or
`steps=1`
(only in Pyre version)
	The maximum and minimum number of time steps for the model, including the 0th time step.

`cpu_limits_in_seconds=`
`360000000`
	Controls the termination of the code based on total wall clock time used. Available only in pure C version.

A.1.7 Initial Conditions

`tic_method=0`	Which method to use to generate the initial temperature field.

`datadir_old="."`
`datafile_old="regtest"`
`solution_cycles_init=0`
`zero_elapsed_time=on`
	If `tic_method=-1`, the initial temperature is read from files. For example, processor #5 will read its initial temperature from old velo file `regtest.velo.5.0` in this case. If `zero_elapsed_time` is `on`, the initial time is set to zero. If it is `off` and `tic_method=-1`, the initial time is read in from the old velo files. Note that this option has no effect when `restart=on`.

`num_perturbations=1`
`perturbmag=0.05`
`perturbl=1`
`perturbm=1`
`perturblayer=5`
 	
`half_space_age=40`
`mantle_temp=1.0`
 	
`blob_center=[-999,-999,-999]`
`blob_radius=0.063`
`blob_dT=0.18`
	If `tic_method=0`. The initial temperature is a linear temperature gradient with perturbations at specific layers, where `num_perturbations` specifies the number of perturbations, and `perturblayer` specifies the layers to be perturbed, representing the number of the mesh node in radial direction. There must be as many entries as `num_perturbations` in a comma-separated list. The perturbation added to each layer is given by: $$mag \times \cos(m\phi) \times P_{lm}(\cos\theta)$$ for the full sphere, and by: $$mag \times \cos(\frac{(\theta-\theta_{min})\, l\pi}{\theta_{max}-\theta_{min}}) \times \cos(\frac{(\phi-\phi_{min})\, m\pi}{\phi_{max}-\phi_{min}})$$ for the regional sphere.
	If `tic_method=1`, T is `1` everywhere, except a cold thermal boundary layer at the top, whose temperature is determined by the half-space cooling model and `half_space_age` (in million of years, Myrs).
	If `tic_method=2,` T is `mantle_temp` everywhere, except for a warm spherical blob and a cold thermal boundary layer at the top, whose temperature is determined by the half-space cooling model and `half_space_age` (in Myrs). The location of the blob is default to the center of the computational domain.
	If `tic_method=3`, the initial temperature is a conductive profile with perturbations to all layers. The perturbation is given by: $$mag \times \sin\left(\frac{(r-r_{in})\pi}{r_{out}-r_{in}}\right)(\sin(m\phi)+\cos(m\phi))\, P_{lm}(\cos\theta)$$ for the full sphere, and by: $$mag \times \sin\left(\frac{(r-r_{in})\pi}{r_{out}-r_{in}}\right) \times \cos(\frac{(\theta-\theta_{min})\, l\pi}{\theta_{max}-\theta_{min}}) \times \cos(\frac{(\phi-\phi_{min})\, m\pi}{\phi_{max}-\phi_{min}})$$ for the regional sphere.
	If `tic_method=4`, the initial temperature is read from grd files.
	If `tic_method=10,` T is `mantle_temp` everywhere, except for a cold thermal boundary layer at the top and perturbations at all layers, similar to `tic_method=3`.
	If `tic_method=11,` T is `mantle_temp` everywhere, except for a hot thermal boundary layer at the bottom and perturbations at all layers, similar to `tic_method=3`.
	If `tic_method=12,` T is `mantle_temp` everywhere, except for a cold thermal boundary layer, a hot thermal boundary layer at the bottom and perturbations at all layers, similar to `tic_method=3`.
	If `tic_method=90,` T is `0` everywhere, except for a single perturbation at the middle layer. This initial temperature is good for comparison with analytical solutions.

A.1.8 Boundary Conditions

`topvbc=0`
`topvbxval=0.0`
`topvbyval=0.0`
	Surface velocity boundary condition parameters. If `topvbc` is 0, `topvbxval` and `topvbyval` specify the tangential surface stress (stress BC). If `topvbc` is 1, `topvbxval` and `topvbyval` specify the tangential surface velocity (velocity BC). The surface normal velocity is zero in these two cases (impermeable BC).

`botvbc=0`
`botvbxval=0.0`
`botvbyval=0.0`
	As above, but for bottom velocity boundary conditions.

`side_sbcs=off`	Enable traction boundary condition for the sidewalls or not. Must be `on` for coupled model.
`pseudo_free_surf=off`	Enable pseudo free surface or not.

`toptbc=1`
`toptbcval=0.0`
	Surface temperature boundary conditions. If `toptbc` is 0, `toptbcval` specifies the surface heat flux (not working in this version). If `toptbc` is 1, the `toptbcval` specifies the surface temperature.

`bottbc=1`
`bottbcval=1.0`
	As above, but for bottom temperature boundary conditions.

`temperature_bound_adj=off`
`depth_bound_adj=0.157`
`width_bound_adj=0.08727`
`lith_age_depth=0.0314`
	Additional parameters for temperature boundary conditions when `lith_age` is `on`.

A.1.9 Non-Dimensional Numbers

`rayleigh=1.0e+5`	This specifies the Rayleigh number, which is one of the most important parameters you may want to change.
`dissipation_number=0.0`	The dissipation number.
`gruneisen=0.0`	The Gruneisen parameter. When this parameter is 0, the code treats it as infinity (i.e., incompressible case).
`surfaceT=0.1`	The non-dimensional value of surface temperature.
`Q0=0.0`	This specifies the internal heating number.

A.1.10 Depth Information

`z_lith=0.014`
`z_410=0.06435`
`z_lmantle=0.105`
`z_cmb=0.439`
	These specify the non-dimensional depth of the Moho, 410km discontinuity, 660km discontinuity and D". These parameters are used to determine the depth of viscosity layers and phase changes (see next two sections). The names are only suggestive. You are free to refer `z_lith` to an arbitrary depth, for example.

A.1.11 Viscosity

`Viscosity=system`	This parameter must be set as indicated.
`visc_smooth_method=3`	This specifies which method to use to smooth viscosity projection for the multigrid solver.
`VISC_UPDATE=on`	If `VISC_UPDATE` is `on`, viscosity will be updated every time step. Otherwise, viscosity will be time-independent.

`num_mat=4`	This specifies the number of material layers. Material 1 is at depth between 0 and `z_lith`, material 2 between `z_lith` and `z_410`, material 3 between `z_410` and `z_lmantle`, and material 4 between `z_lmantle` and the bottom. If `mat_control` is on, then a multiplicative factor is applied to the viscosity, as defined below.
`visc0=1,1,1,1`	The pre-exponent factor of layered viscosity structure (η_0 in the equations below). There must be as many entries as `num_mat` in a comma-separated list.
`TDEPV=off`	Enable temperature dependence or not.
`rheol=3`	When `rheol`=1, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(E_\eta(T_\eta - T))$ When `rheol`=2, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(-T/T_\eta)$ When `rheol`=3, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(\frac{E_\eta}{T^*+T_\eta} - \frac{E_\eta}{1+T_\eta})$, where $T^* = \min(\max(T,0),1)$. When `rheol`=4, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(\frac{E_\eta+Z_\eta(1-r)}{T^*+T_\eta})$ When `rheol`=5, same as `rheol`=3, except the viscosity cut-off is applied before `mat_control`. When `rheol`=6, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(E_\eta(T_\eta - T^*) + Z_\eta(1-r))$ When `rheol`=7, temperature-dependent viscosity is computed by $\eta = \eta_0 \times \exp(\frac{E_\eta+Z_\eta(1-r)}{T+T_\eta} - \frac{E_\eta+Z_\eta(1-r_{inner})}{1+T_\eta})$, where r_{inner} is the inner radius of the mesh. When `rheol`=8, same as `rheol`=3, except viscosity reduction is applied when the temperature exceeds the solidus. When `rheol`=9, same as `rheol`=3, except using T, not T^*. When `rheol`=10, same as `rheol`=8, except using T, not T^*.
`viscE=1,1,1,1` `viscT=1,1,1,1` `viscZ=1,1,1,1`	Parameters defining viscosity law (E_η, T_η, and Z_η in the equations above, respectively). There must be as many entries as `num_mat` in a comma-separated list.
`SDEPV=off`	Enable stress dependence (non-Newtonian) or not.
`sdepv_expt=1,1,1,1` `sdepv_misfit=0.001`	If `SDEPV` is on, these specify the exponent in the viscosity law and the criterion of convergence test. There must be as many entries as `num_mat` in a comma-separated list.
`PDEPV=off`	Pseudo-plastic rheology, implemented by adding a plastic viscosity $\eta_p = \frac{\sigma_y}{(2\epsilon_{II}+10^{-7})+\eta_p^0}$ where the yield stress is defined as $\sigma_y = \min\left(a + b\left(1-z\right), y\right)$ and ϵ_{II} is the second shear strain rate invariant (all non-dimensional). The effective viscosity η' is computed from η and η_p as $\eta' = \frac{\eta_p \eta}{(\eta+\eta_p)}$ (pdevp_eff=on) or $\eta' = \min\left(\eta, \eta_p\right)$ (pdepv_eff=off).
`pdepv_a=1e20,1e20,1e20,1e20` `pdepv_b=0,0,0,0` `pdepv_y=1e20,1e20,1e20,1e20`	Parameters for σ_y.
`pdepv_eff=on`	Effective or minimum viscosity "plasticity" (see above).
`pdepv_offset=0`	Offset for plastic viscosity η_p^0.
`CDEPV=off`	Compositionally dependent viscosity pre-factor (only for `tracer=on`). Will assign two pre-multipliers for C = 0 and C = 1 and compute a geometric mean for all elements depending on the nodal composition. Only works for two flavors and assuming C varies between $[0;1]$.
`cdepv_ff=1,1`	Flavor-wise viscosity pre-factors for C = 0 and C = 1, respectively.

`low_visc_channel=off`
`low_visc_wedge=off`
	Used in conjunction with tracers. The tracers define the upper boundary of the subducted slab.

`lv_min_radius=0.9764`
`lv_max_radius=0.9921`
`lv_channel_thickness=0.0047`
`lv_reduction=0.5`
	If `low_visc_channel` or `low_visc_wedge` is on, this specifies the radial extents of the low viscosity zones and the viscosity reduction factor, respectively.

`VMIN=off`
`visc_min=0.001`
	If `VMIN` is on, minimum viscosity is cut off at `visc_min`.

`VMAX=off`
`visc_max=1000`
	If `VMAX` is on, maximum viscosity is cut off at `visc_max.`

A.1.12 Phase Change Information

`Ra_410=0.0`
`clapeyron410=0.0235`
`transT410=0.78`
`width410=0.0058`
	These specify the phase change parameters (phase change Rayleigh number, Clapeyron slope, ambient temperature, and phase change width, respectively). The depth of this phase change is specified by `z_410`.

`Ra_670=0.0`
`clapeyron670=-0.0235`
`transT670=0.875`
`width670=0.0058`
	As above. The depth of this phase change is specified by `z_lmantle`.

`Ra_cmb=0.0`
`clapeyroncmb=-0.0235`
`transTcmb=0.875`
`widthcmb=0.0058`
	As above. The depth of this phase change is specified by `z_cmb`.

A.1.13 Momentum Equation Solver Parameters

`stokes_flow_only=off`	If you wish only to solve for the velocity once (e.g., Stokes flow) then change this parameter to `on`. However, if you want to do a convection problem, this should be `off`.

`remove_rigid_rotation=on`
`remove_angular_momentum=on`
	Whether to remove rigid body rotation component or net angular momentum from the velocity solution. Only effective for global models.

`Solver=cgrad`	Can be either `cgrad` for conjugate gradient solver or `multigrid` for multigrid solver for the outer loop of the momentum solver.
`node_assemble=on`	Whether to assemble stiffness matrix at the node level or not.

`mg_cycle=1`
`down_heavy=3`
`up_heavy=3`
`vlowstep=1000`
`vhighstep=3`
`max_mg_cycles=50`
	Parameters for `multigrid` and `cgrad` solvers. For `multigrid` solver: `mg_cycle=1` for V cycle and 2 for W cycle; `down_heavy` and `up_heavy` are the number of smoothing passes for downward/upward smoothing; `vlowstep` and `vhighstep` are the number of smoothing passes at lowest/highest levels; `max_mg_cycles` is the maximum number of multigrid cycles used per solve; all these parameters should be small integers.
	For `cgrad` solver, `vlowstep` is the maximum iterations of the conjugate gradient solver and should be a large integer.

`piterations=1000`	Maximum iterations of the outer loop for the momentum solver.
`accuracy=1.0e-4`	Convergence criterion for the momentum solver.

`uzawa=cg`
`compress_iter_maxstep=100`
	Can be either `cg` or `bicg`. If set to `cg`, additional parameters control the maximum iterations.

`precond=on`	Whether to use the preconditioner.

`aug_lagr=on`
`aug_number=2.0e3`
	Whether to use augmented stiff matrix and the weight of the augmented stiff matrix.

A.1.14 Energy Equation Solver Parameters

`ADV=on`	If on, solves the energy equation.
`fixed_timestep=0.0`	If it is equal to 0, the size of the time step is variable and is determined dynamically. Otherwise, the size of the time step is fixed at the specified value.
`finetunedt=0.9`	Set the size of the time step to the specified fraction of a maximum stable advection time step. Must be between 0 and 1.
`adv_sub_iterations=2` `adv_gamma=0.5`	The number of iterations and the weight of each iteration for the energy predictor-corrector solver.
`filter_temp=off`	Using a Lenardic filter to remove the overshoots and undershoots of the temperature field or not. The filter conserves the total energy.
`monitor_max_T=on`	If on, the maximum value of the current and previous temperature fields are compared. If the maximum temperature increases more than 5%, the energy equation solver is rerun with a smaller time-step size. Keep this parameter on to prevent numerical instability.
`inputdiffusivity=1`	Currently, don't change this parameter. It is used only in problems which are integrated backward in time.

A.1.15 Age Information

`start_age=40.0`	Set initial age (in Myrs). This age determines which files of various time-dependent input to read in.
`reset_startage=off`	If `on`, the initial age is set to `start_age`. If it is `off` and `tic_method=-1`, the initial age is read in from previous output.

A.1.16 Debugging Information

`DESCRIBE=off` `BEGINNER=off` `VERBOSE=off`	These parameters affect the echo behavior of the CitcomS parser. Only in non-Pyre version.
`verbose=off`	This is used for debugging. If verbose is `on`, additional information is output to a `.info` file.
`see_convergence=on`	If `on`, the velocity residual will be output on the screen for every iteration of the momentum solver.

A.1.17 HDF5 Output Parameters

`cb_block_size=1048576` `cb_buffer_size=4194304`	Size for collective buffer in MPI-IO.
`sieve_buf_size=1048576`	Size of data sieve buffer.
`output_alignment=262144` `output_alignment_threshold=524288`	Memory alignment.
`cache_mdc_nelmts=10330` `cache_rdcc_nelmts=521` `cache_rdcc_nbytes=1048576`	Cache size for chunked dataset.

A.1.18 Tracer Parameters

`tracer=off`	If on, enables the tracers.
`tracer_ic_method=0` `tracers_per_element=10` `tracer_file="tracer.dat"`	Specify the initial location of the tracers. If `tracer_ic_method=0`, the tracers are generated randomly, with the number of tracers per element specified by `tracers_per_element`. If `tracer_ic_method=1`, the location of the tracers is read from a file specified in `tracer_file`. If `tracer_ic_method`=2, the location of the tracers is read from the old tracer output, similar to `tic_method=-1`.
`tracer_flavors=0`	The number of flavors among the tracers. If set to 0, flavor counting is turned off.
`ic_method_for_flavors=0` `z_interface=0.7`	Specify the initial flavors of the tracers. Used only when `tracer_ic_method=0`. Currently only `ic_method_for_flavors=0` is supported. A layered structure is generated. `z_interface` must be a comma-separated list with `(tracer_flavors-1)` entries. Tracers above the 1st radius in `z_interface` are of flavor 0, between the 1st and 2nd radii are of flavor 1, ... etc.
`regular_grid_deltheta=1.0` `regular_grid_delphi=1.0`	The grid spacing of the regular grid. The regular grid is an auxiliary grid to help locate the tracers.
`chemical_buoyancy=on`	If on, enables thermo-chemical convection.
`buoy_type=1`	If `buoy_type=1`, the composition field is determined by the ratio method [?]. (No other methods are implemented yet.)
`buoyancy_ratio=1.0`	The ratio of chemical density anomaly to the reference density, must be a comma-separated list with `(tracer_flavors-1)` entries.
`itracer_warnings=on`	The warning level of the tracer module.
`tracer_enriched=off` `Q0_enriched=0.0`	Whether the composition anomaly is associated with radioactive heating anomaly. If `on`, specifies the internal heating number inside the composition anomaly.

A.1.19 Dimensional Information

`radius=6371e3` `density=3340.0` `thermdiff=1.0e-6` `gravacc=9.81` `thermexp=3.0e-5` `refvisc=1.0e+21` `cp=1200` `density_above=1030.0` `density_below=6600.0`	Various dimensional information in SI units.

A.1.20 Required Information

`Problem=convection` `Geometry=sphere`	For this version of CitcomS, all of these parameters must be set as indicated.

A.2 CitcomS Facilities and Properties

This section lists the facilities and properties in the Pyre version of CitcomS. Most of the properties have names which are identical to the parameters used in the non-Pyre version of CitcomS and are explained in the section above. This section highlights those which have changed and those which are entirely new. Parameters are given with their default values.

A.2.1 Top-Level Facilities and Properties

`steps=1`	How many time steps to run.
`launcher=mpich`	A facility specifying which launcher to use. Choices include `mpich` and `lam-mpi`.
`scheduler=none`	A facility specifying which scheduler to use. Choices are `lsf`, `pbs`, and `globus`.
`controller`	A facility. User cannot change it.
`solver=regional`	A facility specifying which solver to use. Must be either `regional` or `full`.

A.2.2 `launcher`

The launcher facility controls how CitcomS, an MPI application, is executed on multiple processors. It is the equivalent to `mpirun` in MPI.

`dry=off`	If on, prints the `mpirun` command line and exits without launching the job.
`nodegen`	A printf-styled format string, used in conjunction with `nodelist` to generate a list of machine names. Example: `n%03d`
`nodelist`	A comma-separated list of machine names in square brackets. Example: `[101-103,105,107]`
`command`	`The command used to launch the job. The default value should be customized if using any job scheduler, otherwise the default value should work well.`

A.2.3 `scheduler`

The scheduler facility controls how CitcomS submits jobs to a batch scheduler. Available schedulers include: `none` (default, not using a batch scheduler), `lsf` (for Load Sharing Facility batch scheduler), `pbs` (for Portable Batch System batch scheduler), `sge` (for Sun Grid Engine batch scheduler), and tacc-ranger (a variant of PBS, customized for the Ranger cluster at Texas Advanced Computing Center). Only the command options for these schedulers are listed below. For more information on configuring Pyre for your batch system, see CIG's Pythia page (`geodynamics.org/cig/software/packages/cs/pythia/docs/batch`).

`dry=off`	If `on`, prints the batch script and exits without scheduling the job.
`wait=off`	If `on`, waits for batch jobs to finish before exiting.

A.2.4 `job`

The `job` facility controls options for individual batch jobs.

`name`	The name of the job.
`queue`	The name of the queue to which the job is scheduled.
`walltime`	Time limit for the job, specified using units, e.g., `5*minute`, `2*hour`.
`stdin=/dev/null`	File to read as the input stream.
`stdout=stdout.txt`	File to write for the output stream.
`stderr=stderr.txt`	File to write for the error stream.

A.2.5 `controller`

`monitoringFrequency=100`	The time-step interval between disk output. It replaces the `storage_spacing` parameter in the (old) CitcomS input file.
`checkpointFrequency=100`	The time-step interval between checkpoint output, which can be used later to resume the computation.

A.2.6 `solver`

`mesher`	A facility. Must be either `full-sphere` or `regional-sphere`. User does not need to specify it. It is set by the `solver` facility automatically.
`tsolver`	A facility for the temperature solver using Petrov-Galerkin time integration.
`vsolver`	A facility for the velocity solver using Boussinesq approximation and the Uzawa algorithm.
`bc`	A facility of boundary conditions.
`const`	A facility for dimensional constants.
`ic`	A facility for initial conditions.
`output`	A facility for output options.
`param`	A facility for some additional input parameter files.
`phase`	A facility for phase change parameters.
`tracer`	A facility for tracer parameters.
`visc`	A facility for viscosity parameters.
`datadir="."`	
`datafile="regtest"`	
`datadir_old="."`	
`datafile_old="regtest"`	
`rayleigh=100000`	
`dissipation_number=0.0`	
`gruneisen=0.0`	
`surfaceT=0.1`	
`Q0=0`	
`stokes_flow_only=off`	
`verbose=off`	
`see_convergence=on`	

A.2.7 `solver.mesher`

`nprocx=1`	
`nprocy=1`	
`nprocz=1`	
`coor=0`	
`coor_file="coor.dat"`	
`coor_refine=0.1,0.15,0.1,0.2`	
`nodex=9`	
`nodey=9`	
`nodez=9`	
`levels=1`	
`radius_outer=1`	
`radius_inner=0.55`	
`theta_min=1.0708`	
`theta_max=2.0708`	
`fi_min=0`	
`fi_max=1`	

A.2.8 solver.tsolver

ADV=on	
filter_temp=off	
monitor_max_T=on	
finetunedt=0.9	
fixed_timestep=0.0	
inputdiffusivity=1	
adv_gamma=0.5	
adv_sub_iterations=2	

A.2.9 solver.vsolver

Solver=cgrad	
node_assemble=on	
precond=on	
accuracy=1.0e-4	
uzawa=cg	
compress_iter_maxstep=100	
mg_cycle=1	
down_heavy=3	
up_heavy=3	
vlowstep=1000	
vhighstep=3	
max_mg_cycles=50	
piterations=1000	
aug_lagr=on	
aug_number=2000	
remove_rigid_rotation=on	
remove_angular_momentum=on	

A.2.10 solver.bc

side_sbcs=off	
pseudo_free_surf=off	
topvbc=0	
topvbxval=0	
topvbyval=0	
botvbc=0	
botvbxval=0	
botvbyval=0	
toptbc=1	
toptbcval=0	
bottbc=1	
bottbcval=1	
temperature_bound_adj=off	
depth_bound_adj=0.157	
width_bound_adj=0.08727	

A.2.11 solver.const

radius=6.371e+06	
density=3340.0	
thermdiff=1e-06	
gravacc=9.81	
thermexp=3e-05	
refvisc=1e+21	
cp=1200	
density_above=1030.0	
density_below=6600.0	
z_lith=0.014	
z_410=0.06435	
z_lmantle=0.105	
z_cmb=0.439	

A.2.12 solver.ic

restart=off	
post_p=off	
solution_cycles_init=0	
zero_elapsed_time=on	
tic_method=0	
num_perturbations=1	
perturbl=1	
perturbm=1	
perturblayer=5	
perturbmag=0.05	
half_space_age=40	
mantle_temp=1.0	
blob_center=[-999,-999,-999]	
blob_radius=0.063	
blob_dT=0.18	

A.2.13 solver.output

output_format="ascii"	
output_optional="surf,botm, tracer"	
output_ll_max=20	
self_gravitation=off	
use_cbf_topo=off	
cb_block_size=1048576	
cb_buffer_size=4194304	
sieve_buf_size=1048576	
output_alignment=262144	
output_alignment_threshold= 524288	
cache_mdc_nelmts=10330	
cache_rdcc_nelmts=521	
cache_rdcc_nbytes=1048576	

A.2.14 solver.param

reference_state=1	
refstate_file="refstate.dat"	
mineral_physics_model=3	
file_vbcs=off	
vel_bound_file="bvel.dat"	
mat_control=off	
mat_file="mat.dat"	
lith_age=off	
lith_age_file="age.dat"	
lith_age_time=off	
lith_age_depth=0.0314	
start_age=40	
reset_startage=off	

A.2.15 solver.phase

Ra_410=0	
clapeyron410=0.0235	
transT410=0.78	
width410=0.0058	
Ra_670=0	
clapeyron670=-0.0235	
transT670=0.78	
width670=0.0058	
Ra_cmb=0	
clapeyroncmb=-0.0235	
transTcmb=0.875	
widthcmb=0.0058	

A.2.16 solver.tracer

tracer=off	
tracer_ic_method=0	
tracers_per_element=10	
tracer_file="tracer.dat"	
tracer_flavors=0	
ic_method_for_flavors=0	
z_interface=0.7	
itracer_warnings=on	
regular_grid_deltheta=1.0	
regular_grid_delphi=1.0	
chemical_buoyancy=on	
buoy_type=1	
buoyancy_ratio=1.0	
tracer_enriched=off	
Q0_enriched=0.0	

A.2.17 `solver.visc`

`Viscosity="system"`	
`visc_smooth_method=3`	
`VISC_UPDATE=on`	
`num_mat=4`	
`visc0=1,1,1,1,`	
`TDEPV=off`	
`rheol=3`	
`viscE=1,1,1,1,`	
`viscT=1,1,1,1,`	
`viscZ=1,1,1,1,`	
`SDEPV=off`	
`sdepv_misfit=0.001`	
`sdepv_expt=1,1,1,1,`	
`PDEPV=off`	
`pdepv_a=1e20,1e20,1e20,1e20`	
`pdepv_b=0,0,0,0`	
`pdepv_y=1e20,1e20,1e20,1e20`	
`pdepv_eff=on`	
`pdepv_offset=0`	
`CDEPV=off`	
`cdepv_ff=1,1`	
`low_visc_channel=0`	
`low_visc_wedge=0`	
`lv_min_radius=0.9764`	
`lv_max_radius=0.9921`	
`lv_channel_thickness=0.0047`	
`lv_reduction=0.5`	
`VMIN=off`	
`visc_min=0.001`	
`VMAX=off`	
`visc_max=1000`	

A.2.18 `layout`

The following facilities are only used in the coupled CitcomS solver.

`containing_group=[0-11]`	The processors to be assigned to the csolver.
`embedded_group=[12]`	The processors to be assigned to the esolver.

A.2.19 `ccoupler`

`si_unit=off`	Whether to exchange information in SI units.
`cartesian=off`	Whether to exchange information in Cartesian coordinate.
`two_way_communication=on`	Whether to transfer information from esolver to csolver.
`exchange_initial_temperature =on`	Whether to ensure consistent initial temperature between csolver or esolver.
`exchange_pressure=on`	Whether to send the pressure solution to esolver.

A.2.20 `ecoupler`

`si_unit=off`	Whether to exchange information in SI units.
`cartesian=off`	Whether to exchange information in Cartesian coordinate.
`two_way_communication=on`	Whether to transfer information from esolver to csolver.
`exchange_initial_temperature=on`	Whether to ensure consistent initial temperature between csolver or esolver.
`exchange_pressure=on`	Whether to receive the initial guess of pressure from csolver.
`exclude_top=off`	Whether to exclude the top boundary when receiving boundary conditions from the csolver. If the esolver has imposed plate velocity and constant temperature at the surface, set this to on.
`exclude_bottom=off`	Whether to exclude the bottom boundary when receiving boundary conditions from the csolver.
`amending_outflow=off`	Whether to amend the velocity boundary conditions to satisfy the continuity equation.

A.2.21 `journal`

The Pyre facility `journal` provides five types of debugging output for conceptually different types of information. The journal output stream can be instantiated as separated channels multiple times, and each channel can be individually activated or deactivated, and sent to different locations (the terminal, sockets, files, devices, etc.). The streams and their default states of the journal streams are:

`debug`	Debugging information. Default off.
`error`	Unrecoverable runtime error. Default on.
`firewall`	Fatal programming error. Default on.
`info`	Descriptive information. Default off.
`warning`	Recoverable runtime error. Default off.

For coupled solvers, there is a large amount of debugging information that outputs through the journal facility. That output can be turned on/off with command line options. The most important ones are `journal.debug.Exchanger` and `journal.debug.CitcomS-Exchanger`. Other less important options can be found in `tests/coupled.cfg`. To enable the journal output, try running the scripts with the following options:

```
[CitcomS.journal.debug]
Exchanger = on
CitcomS-Exchanger = on
```

Appendix B

CitcomS Input File Format

B.1 Introduction

CitcomS expects Unix-styled ASCII files (i.e., no carriage character following new line character) for all input files. This can be a nuisance in DOS/Windows systems. You may want to find a text editor that can write Unix-style ASCII files. In the following, words in normal `courier` must be input exactly as shown, while *`italicized`* words should be substituted by your values. All parameters are in non-dimensional units unless specified.

B.2 Coordinate Files

For regional version of CitcomS, the mesh must be regular, but the mesh spacing may be unequal. The `coor_file` has the format:

```
nsd= 1
1      theta1
2      theta2
...    ...
nodex theta_nodex
nsd= 2
1      phi1
2      phi2
...    ...
nodey phi_nodey
nsd= 3
1      r1
2      r2
...    ...
nodez r_nodez
```

For the full spherical version of CitcomS, the mesh of each cap must be regular and equidistant in the horizontal dimension. Only the vertical dimension is specified by `coor_file`. The `coor_file` has the format:

```
nsd= 3
1      r1
2      r2
...    ...
nodez r_nodez
```

B.3 Velocity Boundary Condition Files

If `vel_bound_file` is set to `bvel.dat`, then it is necessary to have one `bvel.dat` file per cap for each million-year interval. For example, if `start_age=83`, then the following files are needed for the regional mesh:

```
bvel.dat84
bvel.dat83
bvel.dat82
...
bvel.dat0
```

> **TIP:** Even though the model starts at 83 million years before the present, by default it will attempt to read in both a file for 84 and a file for 83 million years ago. To deal with this, create a duplicate copy of `bvel.dat83` and call it `bvel.dat84`

In this example, the model age will start at 83 Ma and decrease toward 0 Ma. At any particular age, the velocity boundary conditions are interpolated in time according to the nearest `bvel.dat` files. Once the age becomes negative, the velocity boundary conditions are specified by the content of `bvel.dat0.`

> **TIP:** By setting `start_age=0` and providing identical `bvel.dat0` and `bvel.dat1`, you can effectively get time-invariant velocity boundary conditions.

For the global mesh, each of the 12 caps requires one file for each million-year interval. For example, these files are needed for an 82-million-year age:

```
bvel.dat82.0
bvel.dat82.1
bvel.dat82.2
...
bvel.dat82.11
```

Each velocity boundary condition file has the format:

```
Vx Vy
```

The units of `Vx` and `Vy` are cm/yr.

B.4 Material Files

In this version of CitcomS, the implementation of material support is not working. The material output has been disabled and is documented here for completion. If `mat_file` is set to `mat.dat`, then it is necessary to have one `mat.dat` file for each million-year interval. For example, if `start_age=83`, then the following files are needed:

```
mat.dat84
mat.dat83
mat.dat82
...
mat.dat0
```

The same note about `vel_bound_file` (see Section B.3 above) applies. The format of `mat_file` is:

```
n viscosity_factor
```

B.5 Lithosphere Age Files

If `lith_age_file` is set to `lith.dat`, then it is necessary to have one `bvel.dat` file for each million-year interval. For example, if `start_age=83`, then the following files are needed for the regional mesh:

```
lith.dat84
lith.dat83
lith.dat82
...
lith.dat0
```

The same note about `vel_bound_file` (see Section B.3 above) still applies. The input age is in millions of years. For the global mesh, each of the 12 caps requires one file for each million-year interval. For example, these files are needed for an 82-million-year age:

```
lith.dat82.0
lith.dat82.1
lith.dat82.2
...
lith.dat82.11
```

The format of `lith_age_file` is:

```
n age
```

B.6 Tracer Files

This file contains the initial location of all tracers. The first line is the number of tracers and the number of columns in the file. The first three columns are the coordinates of the tracers. The fourth column, if present, indicates the flavor of the tracers.

```
num_tracers num_columns
theta phi radius [flavor]
```

B.7 Reference State Files

This file contains the profiles of the reference state. This file must contain at least `nodez` lines, and each line must contain 7 columns, where the meaning of each column is:

ρ_r g α c_P *reserved reserved reserved*

Appendix C

CitcomS Output File Format

C.1 Introduction

The format of the output files of CitcomS is described here. In the following sections, the model prefix is assumed as `test-case`, the processor number as 0, and the time step as 10. All outputs are in non-dimensional units unless specified.

C.2 Postprocessed Cap Output (`test-case.cap00.10`)

The command `autocombine.py` produces 1 cap file for regional CitcomS and 12 cap files for the full CitcomS. The first line of the cap file is a comment describing the node geometry (`nodex` × `nodey` × `nodez`). The rest of the file is column-based, where the meaning of each column is:

```
colatitude longitude radius vel_colat vel_lon vel_r temperature viscosity
```

C.3 Postprocessed Opt Output (`test-case.opt00.10`)

If the optional output contains node-based fields (e.g., `comp_nd`, `pressure`, and `stress`), the command `autocombine.py` produces 1 opt file for regional CitcomS and 12 opt files for the full CitcomS. The first line of the opt file is a comment describing the node geometry (`nodex` × `nodey` × `nodez`). The rest of the file is column-based, where the meaning of each column is defined in the corresponding `.general` file. For example, if the `.general` file contains these two lines, then the first column in the opt file is the composition, the 2nd column is the pressure, and the last six columns are the stress.

```
field = composition, pressure, stress
structure = scalar, scalar, 6-vector
```

C.4 Postprocessed Surf Output (`test-case.surf0.10`)

An undocumented (and unmaintained) post-processing script `batchsurf.py` can combine the `coord`, `surf`, and `botm` output. The format of the combined surf file is listed for completion. The first line of the surf file is a comment describing the node geometry (`nodex` × `nodey`). The rest of the file is column-based, where the meaning of each column is:

```
colatitude longitude topography heat_flux vel_colat vel_lon
```

C.5 Time Output (`test-case.time`)

This file reports non-dimensional elapsed model time and spent CPU time (in seconds), and it is only outputted on computer node 0. The meaning of each column is:

```
step total_t delta_t total_cpu_time step_cpu_time
```

C.6 ASCII Output

C.6.1 Coordinate Output (`test-case.coord.0`)

This file is only outputted at the 0th time step. The first line is a header. The rest of the file is column-based, where the meaning of each column is:

```
colatitude longitude radius
```

C.6.2 Velocity and Temperature Output (`test-case.velo.0.10`)

The first two lines of this file are headers. The rest of the file is column-based, where the meaning of each column is:

```
vel_colat vel_lon vel_r temperature
```

C.6.3 Viscosity Output (`test-case.visc.0.10`)

The first line of this file is a header. The rest of the file is column-based, where the meaning of the only column is:

```
viscosity
```

C.6.4 Material Output (`test-case.mat.0`)

In this version of CitcomS, the implementation of material support is not working. The material output has been disabled and is documented here for completion. This file is only outputted at the 0th time step. There is no header. The meaning of each column is:

```
element_number layer material
```

C.6.5 Surface Variables Output (`test-case.surf.0.10` and `test-case.botm.0.10`)

The first line of each file is a header. The rest of each file is column-based, where the meaning of each column is:

```
topography heat_flux vel_colat vel_lon
```

C.6.6 Stress Output (`test-case.stress.0.10`)

The first two lines of the file are headers. The rest of the file is column-based, where the meaning of each column is:

$S_{\theta\theta}$ $S_{\phi\phi}$ S_{rr} $S_{\theta\phi}$ $S_{\theta r}$ $S_{\phi r}$

You can use the above values to form the following symmetric stress tensor:

$$\begin{bmatrix} S_{\theta\theta} & S_{\theta\phi} & S_{\theta r} \\ S_{\phi\theta} & S_{\phi\phi} & S_{\phi r} \\ S_{r\theta} & S_{r\phi} & S_{rr} \end{bmatrix} \tag{C.1}$$

C.6.7 Pressure Output (`test-case.pressure.0.10`)

The first line of the file is a header. The rest of the file is column-based, where the meaning of the only column is:

```
pressure
```

C.6.8 Horizontal Average Output (`test-case.horiz_avg.0.10`)

The first line of the file is a header. The rest of the file is column-based, where the meaning of each column is:

```
radius temperature RMS(V_horizontal) RMS(V_vertical)
```

C.6.9 Geoid Output (`test-case.geoid.10`)

The first line of the file is a header. The rest of the file is column-based, where the meaning of each column is:

col 1	spherical harmonic degrees (`l`)
col 2	spherical harmonic orders (m)
col 3	cosine coefficients of total geoid ($\mathrm{S_{lm}}$)
col 4	sine coefficients of total geoid ($\mathrm{C_{lm}}$)
col 5	cosine coefficients of the geoid due to surface topography
col 6	sine coefficents of the geoid due to surface topography
col 7	cosine coefficients of the geoid due to internal density variation
col 8	sine coefficients of the geoid due to internal density variation

The units of the geoid coefficients are meters. The geoid field can be reconstructed by

$$\sum \left(S_{lm}\sin(m\phi) + C_{lm}\cos(m\phi)\right) P_{lm}(\cos\theta)$$

where P_{lm} is the associated Legendre polynomials.

C.6.10 Tracer Output (`test-case.tracer.0.10`)

The first line of the file is a header. The second field in the header is the number of tracers in the file. The third field in the header is the number of columns in the file. The rest of the file is column-based, where the meaning of the columns is:

```
theta phi radius [flavor]
```

C.6.11 Composition Output (`test-case.comp_el.0.10` and `test-case.comp_nd.0.10`)

These files contain the composition field either on the element (`comp_el`) or on the node (`comp_nd`). The format of the files is the same. The first two lines of the file are headers. The fourth field in the first line is the number of composition components. In the second line, the first field is the current bulk sum of composition 1. The second field is the initial bulk sum of composition 1. The third and fourth fields (if they exist) are the current and initial bulk sum of composition 2, etc. The rest of the file is column-based, where the meaning of the columns is:

```
composition1 [composition2 ... compositionN]
```

C.6.12 Heating Output (`test-case.heating.0.10`)

These files contain various heating terms on the element. The first two lines of the file are the headers. The rest of the file is column-based, where the meaning of the columns is:

```
adiabatic_heating viscous_heating latent_heating
```

C.7 HDF5 Output (`test-case.h5`)

The format and layout of HDF5 output is described in Section 4.4 on page 43.

C.8 Misc. Binary Output

C.8.1 Checkpoint Output (`test-case.checkpoint.0.10`)

These files are used for restarting the run. These files are not portable. You cannot run a model on machine A, copy the checkpoint files to machine B, and expect you can always restart the simulation on machine B. Their format is undocumented on purpose and will remain so.

C.8.2 Domain Output (`test-case.domain`)

This file is only outputted at the 0th time step. This file is output by rank-0 processor only. It contains the domain bounds of all processors in binary format. The file begins with four intergers: `nproc`, `ncolumns`, `const1`, `const2`, where `nproc` is the number of processors, `ncolumns` is 10, `const1` and `const2` are used to detect binary incompatibility. The rest of the file has `nproc`×`ncolumns` doubles, the columns contains the min/max of radius and the θ, ϕ coordinates of the four corners:

```
r_min r_max theta0 phi0 theta1 phi1 theta2 phi2 theta3 phi3
```

C.8.3 Coordinate Binary Output (`test-case.coord_bin.0`)

These files are only outputted at the 0th time step. These files contain the coordinates of nodes in binary format. The file begins with four intergers: `nodex`, `nodey`, `nodez`, `const`, then followed by the θ, ϕ, r coordinates of all nodes.

C.8.4 Seismic Output (`test-case.seismic.0.10`)

These files contain the density, P wave velocity and S wave velocity in binary format. The density of all nodes is stored in double, then followed by P wave velocity of all nodes, and finally S wave velocity of all nodes.

Appendix D

License

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. Copyright the software, and

2. Offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

NO WARRANTY

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found. For example:

> One line to give the program's name and a brief idea of what it does. Copyright © (year) (name of author)
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright © year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items – whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> (signature of Ty Coon)
> 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography